AUGUST 3, 1998 COMPUTERWORLD

A Y2K pioneer seeks (and deserves) recognition

Bill Schoen was passionate about the problem long before the world was ready to listen

By Paul Gillin

Opinion, Aug. 3, 1998 It was a frigid Wednesday in February 1984, and I was stumped for a story idea. The software business was dead that time of year — not good news to a guy who wrote about software for Computerworld.

Then the phone rang. On the other end was a guy from Detroit who had an idea that was so offbeat, so screwy, that I thought it just might make my editors happy. It was good for a laugh, at least.

Bill Schoen had this wacky idea that a lot of computers — I mean a lot of them — would stop working on New Year's Day 2000. Schoen wasn't just curious about this problem, he was passionate about it. He had even formed a one-man consulting company to try to spread the gospel. But no one, he said, wanted to listen.

I listened. Heck, I was desperate. I interviewed Schoen, talked to one of his clients and wrote it up. My editors put the story on page 7 of the Feb. 13, 1984, issue. It was the first article to appear in a major publication about the year 2000 problem (see the original story.)

I spoke to Bill Schoen again the other day. Now 51, he's still in Detroit, still programming and a little amazed about how prophetic his crusade was. Though he wishes he could have had a piece of the five-figure speaking fees that top year 2000 consultants make, what Schoen really wants today is a little recognition.

Schoen first stumbled on the year 2000 problem in 1983 while programming in the basement of a Big Three automaker. It wasn't rocket science; data processing people had known about the risk since the 1970s. It's just that no one thought their code would last that long.

Schoen thought it might, and a little analysis of his employer's code library proved to him that software had considerably more staying power than many people thought. If business had started coding for the millennium in 1983, "95% of the problems wouldn't be there today," he says wistfully.

For Schoen, the year 2000 problem became a mission. He coded up a little Cobol routine on his Commodore 64 that solved the problem. He named his company Charmar Enterprises and created the Charmar Correction, a cure for "the serious problem ignored by the entire data processing community."

He was an army of one. Booted out of the CIO's office in some of the biggest companies in America, Schoen landed only two sales for the Charmar Correction. He folded Charmar's tent in 1984 and went back to programming. "I've never been able to rise above the level of working stiff," he says.

AHEAD OF HIS TIME

Schoen's story is about missed opportunity and being too far ahead of the curve. The New York Times called him in 1988 to ask about the year 2000 problem, but Schoen was sick of rejection and asked the reporter not to quote him. His efforts to launch a year 2000 consultancy last year fizzled when the client's CEO died.

Now he's got a Web site that tells a little of the story of the guy who had an answer long before anyone else was asking the question. Visit Bill Schoen's Web site. Send him an E-mail. This guy was really on to something.

Copyright 1998 Computerworld Inc., Framingham MA.
Reprinted by permission of Computerworld.

FEBRUARY 13, 1984 COMPUTERWORLD 7
NEWS

The problem you may not know you have

There is a bug in every Cobol program in your library.
You probably don't know about it, and you almost certainly haven't had to worry about it yet. But when it shows up, it will hit your entire shop, reducing data entry procedures to a mush of error messages.

By Paul Gillin
CW staff

NOVI, Mich. – When systems analyst William Schoen has presented DP managers with that pitch, he has understandably piqued their interest. But he has had trouble getting them to take his case seriously when they find out what the bug is.

The problem is the year 2000. The turn of the calendar presents a procedural issue that is acknowledged occasionally in trade conference jokes and DP shop banter but has attracted little serious attention in the industry.

The root of the issue is the industrywide standard of using two-digit year date fields in Cobol programs. Error-checking procedures typically rely upon dates proceeding sequentially, with one year's date being greater than that of previous years.

However, programs will be thrown for a loop after the turn of the century when they have to cope with the two-digit date field "00" being greater than "99." Schoen ticked off the problems that will arise if the issue is not dealt with: "Program logic is going to malfunction all over the place. Sequential data processes are going to abend. A great many on-line modules won't accept the new dates because they include sequence checks. Most sorts won't work, and a great many literals won't work."

Perhaps understandably, the issue has received little serious attention from a DP community that is concerned with more immediate problems. But Schoen claims data processing should start paying attention now to the problem of making programs "year 2000-compatible" in order to avoid headaches when the time of reckoning draws near.

He rationalizes that most large firms maintain a library of thousands of Cobol programs, most of which have at least one date field. In addition, many of those programs have literals, sequence processes and error checks interspersed throughout, meaning that they will require modifications in several places.

"You can't assume every date field has the word 'date' in it," Schoen said. "You also can't assume every field that looks like a date field is a date field. You're going to have to look at every program and figure it out."

For that reason, he said, the "black box" approach of simply running every program through a modification routine is impractical. Some date fields are bound to be missed. The only real solution is to write all new programs to be year 2000-compatible.

Not surprisingly, Schoen has developed a method to do that. The Charmar Correction is a package consisting of two Cobol subroutines that can be inserted into new programs to resolve the problem. The $995 purchase price also includes an analysis of the problem and a methodology to deal with it.

Programs made to work

"They make programs work right, and they include directions to make sorts simple," he said.

Schoen has done some calculations to show why it makes it makes sense to tackle the problem today. He figures it costs $300 to modify a program, and there is a minimum of 50 programs per programmer in the average corporate library. Assuming that half of those programs will need to be modified, that comes to a cost of $7,500 per programmer if the conversion is left to the last minute.

Based on scans of existing libraries, Schoen has decided that if firms begin implementing the changes now, by the year 2000 less than 2% of their programs will require modification. If they wait for just six years to begin the process, the figure balloons to 25% to 30% of their library.

"Why should companies continue to write software that is not year 2000-compatible when it's just as easy to do it the right way now?" he reasons. Not many DP managers have bought this argument so far. He has been escorted out of buildings by security officers more than once, Schoen said.

Manager gets 16-year jump on Cobol bug

TOLEDO, Ohio – Although he agrees that making Cobol programs "year 2000-compatible" is not an issue of immediate concern, a DP manager here has elected to become the first user of a set of subroutines designed to make the transition to the new millennium a smooth one.

Michael Ehrick, director of computer services at Libbey-Owens-Ford, Inc., became interested in preparing Cobol programs for the inevitable march of time after he received a direct mail advertisement from William Schoen, a systems analyst based in Novi, Mich.

Schoen's Charmar Enterprises, Inc. markets a product that addresses the problems that may emerge when the year 2000 arrives and plays havoc with date fields in existing Cobol programs (see related story). His product is "a subroutine that does some elegant arithmetic routines that let you compare date fields with two-digit years in such a way that the year '00' is greater than the year '99,'" Ehrick said. Schoen also offers an approach to sorting so that the same end is achieved, according to Ehrick.

Ehrick put Schoen in contact with Libbey-Owens-Ford's manager of application development. The company expects to begin including the subroutine in new applications as soon as the next development project is undertaken.

The subroutines are "nothing an advanced computer science major couldn't do," Ehrick said. But he added, "a lot of creative thought went into this, along with a lot of dogmatism."

Copyright 1984 Computerworld Inc., Framingham MA.
Reprinted by permission of Computerworld.

4/9/1 (Item 1 from file: 15)
DIALOG(R)File 15:ABI/Inform(R)
(c) 2000 Bell & Howell. All rts. reserv.

00761420 94-10812
Doomsday 2000
De Jager, Peter
Computerworld v27n36 PP: 105-109 Sep 6, 1993 CODEN: CMPWAB ISSN: 0010-4841 JRNL CODE: COW
DOC TYPE: Journal article LANGUAGE: English LENGTH: 3 Pages
WORD COUNT: 1415

ABSTRACT: The information systems (IS) community is heading toward the year 2000 and the failure of its standard date format: MM/DD/YY. The problem is twofold - the date issue itself and, more important, the reluctance of IS to address the problem. IS must identify and correct all the date data and check the integrity of all calculations involving date information. Since there are few standards for labeling data used in date calculations, IS will have to examine each line of code and make the necessary changes. It is estimated that Fortune 50 organizations will need to spend $50 million-$100 million each to convert all their existing systems to accept the change from the year 1999 to 2000. The inability of the IS industry to even think about such a project is troublesome.

TEXT: Have you ever been in a car accident? Time seems to slow down as you realize you're going to crash into the car ahead of you.

It's too late to avoid it--you're going to crash. All you can do now is watch it happen.

The information systems community is heading toward an event more devastating than a car crash. We are heading toward the year 2000. We are heading toward a failure of our standard date format: MM/DD/YY.

Unfortunately, unlike the car crash, time will not slow down for us. If anything, we're accelerating toward disaster.

This is a good news/bad news story. First the bad news: There is very little good news. There is no way to avoid the fact that our information systems are based on a faulty standard that will cost the worldwide computer community billions of dollars in programming efforts.

Perhaps more importantly, we are going to suffer a credibility crisis. We and our computers were supposed to make life easier; this was our promise. What we have delivered is a catastrophe.

The problem is twofold: the date issue itself and, more importantly, our reluctance to address the problem.

PROBLEM ID

What exactly is the "problem"? To save storage space--and perhaps reduce the amount or Keystrokes necessary to enter a year--most IS groups have allocated two digits to the year. For example, "1993" is stored as "93" in our data files, and "2000" will be stored as "00." These two-digit dates exist on millions of data files used as input to millions of applications.

This two-digit date affects data manipulation, primarily subtractions and comparisons. For instance, I was born in 1955. If I ask the computer to calculate how old I am today, it subtracts 55 from 93 and announces that I'm 38.

So far so good. But what happens in the year 2000? The computer will subtract 55 from 00 and will state that I am-55 years old. This error will affect any calculation that produces or uses time spans, such as an interest calculation.

If you have some data records and want to sort them by date (e.g., 1965,

1905, 1966), the resulting sequence would be 1905, 1965, 1966. However, if you add in a date record such as 2015, the computer, which reads only the last two digits of the date, sees 05, 15, 65, 66 and sorts them incorrectly.

These are just two types of calculations that are going to produce garbage. There are others.

The task facing us is to identify and correct all the date data and check the integrity of all calculations involving date information. We must correct the data residing in all data files or write code to handle the problem.

THE STARTING POINT

How do we identify the problem data and the associated calculations? We have few, if any, standards for labeling data used in date calculations. The only choice we have is to examine each line of code and make the necessary changes.

One IS person I know of performed an internal survey and came up with the following results: Of 04 systems, 18 would fail in the year 2000. These 18 mission-critical systems were made up of 8,74 programs and data-entry screens as well as some 3,313 databases. With less than seven years to go, someone is going to be working overtime.

By the way, this initial survey required 10 weeks of effort. Ten weeks just to identify the problem areas.

How many systems do you have? How many lines of code do you have in you organization? How many data files? How many maintenance programmers?

The problem extends beyond mere calculations and into the I/O processes of every application. Can you enter 2000 into your data screen, or can you enter only two digits, forcing the input of 00? Can your hard-copy reports print four digits?

The crisis is very real and potentially very costly. Ken Orr, principal at the Ken Orr Institute, and Larry Martin, president of Data Dimensions, Inc. estimate that Fortune 50 organizations will each have to spend about 35 to 40 cents per line of code to convert all their existing systems to accept the change from the year 1999 to 2000.

That translates into about $50 million to $100 million for each company. The mind boggles at a maintenance 1.2 problem with that price tag.
And the costs could be even higher. "The truth is, until we work through a complete cycle with some large organization, we are not going to really know," Orr says.

I have spoken at association meetings and seminars, and when I ask for a show of hands of people addressing the problem, the response is underwhelming. If I get one in 10 respondents, I'm facing an enlightened group.

Typically, all I get are snickers and comments such as, "I won't be in this position or this company in the year 2000. It's not my problem."

This attitude in the computing community is the real problem. It is very difficult for us to acknowledge that we made a "little" error that will cost companies millions of dollars. It is also a "pay me now or pay me later" situation.

"We in the IS industry have not been paying our way," says Gerald Weinberg, author of Quality Software Management and winner of the 1991 J. D. Warnier Prize for Excellence in Information Science. "We have been building up a 'national debt' just as surely as the U.S. has been building up a money debt. It will be paid by our children--our successors-one way or another," Weinberg says.

We don't have a choice. We must start addressing the problem today or there won't be enough time to solve it. Status quo means applications that will produce meaningless results in the new millennium.

Weinberg says he believes. this procrastination is an indication of deep management malaise. "If software engineering managers cannot manage a change that they've had 1,000 years to prepare for, how can we expect them to manage a change that happens without notice? In other words, if this change causes a crisis in your organization, everything will cause a crisis in your organization--and often nothing will cause a crisis."

The inability of the industry to even think about such a project is troublesome. "No one wants to step up to the issue--not [IS] management, not the vendors, not the industry gurus," Orr says. "As with all legacy systems, this problem is messy, expensive and unromantic. No one wants to o in and tell management they have a multimillion-dollar requirement just to keep the business running and that they really have no options."

The reason nothing is being done, says Capers Jones, chairman at Software Productivity Research, Inc., is that the software industry isn't used to taking long-term preventative steps. "I expect that most companies will not start worrying about the problem until 1999," Jones says. "For some, this will be too late."

NOW THE GOOD NEWS

There is good news. Object-oriented systems may be able to help. Faced with the huge maintenance costs of fixing their systems, firms may opt to rewrite systems from scratch using object-oriented programming techniques. Tom Love, IBM vice president of the Object-Oriented Group, is a proponent of this theory.

Some companies are unveiling testing and inventory tools that may ease the identification of trouble spots.

Others are hoping that bombarding people with information is the best remedy. To that end, William Goodwin in Brooklyn, N.Y., publishes a newsletter entitled "Tick, Tick, Tick," which brings together people in the IS industry concerned about the impact of the year 2000.

But is the warning falling on deaf ears?
"I feel like a lone voice crying in the wilderness," says Brian Pitts, one of Goodwin's subscribers and project manager at Berry Co. in Dayton, Ohio. "Current economic conditions are making this problem more difficult to address. Management is focused on short-term results and is placing long-term negative consequences on the back burner."

The next seven years will be filled with dire predictions. "You are going to become very, very tired of millennium moaners telling you that your software will fail as it enters the new millennium," says Nicholas Zvegintzov, publisher of Software Maintenance News. "But be patient with them. There really is something to be said for them."

De Jager is an industry speaker on the topics of change, creativity and management of technology. He can be reached at (416) 792-8706 or via CompuServe (70611,2576) and MCI Mail (PDEJAGER).

THIS IS THE FULL-TEXT. Copyright CW Publishing Inc 1993

DESCRIPTORS: Calendars; Time; Problems; Computer programming; Contingency planning; Information systems
CLASSIFICATION CODES: 5240 (CN=Software & systems); 2310 (CN=Planning)

8/9/2 (Item 2 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)
(c) 2000 The Gale Group. All rts. reserv.

01806334 SUPPLIER NUMBER: 17112198 (THIS IS THE FULL TEXT)
DateServer 2000. (Computer Software Corp prepares legacy **systems for year 2000)**
Roberts, Chris
Enterprise Systems Journal , v9, n11, p10(1)
Nov, 1994
ISSN: 1053-6566 LANGUAGE: English RECORD TYPE: Fulltext; Abstract
WORD COUNT: 613 LINE COUNT: 00053

ABSTRACT: Computer Software's $10,000 DateServer 2000 software, available for MVS or VSE, offers a solution to the problem of correcting multicentury dates. The year 2000 will present unique problems, especially in applications that make date field comparisons using date field values with two digits. One approach is to expand the date field to four digits, but this is a costly solution and requires converting the date fields, input screens and files and re-coding existing programs. DateServer 2000 eliminates the need for any conversion. Programmers simply use the CALL statement to access DateServer routines to make field comparisons or calculations. When the program executes, DateServer will assign the appropriate century value based on installation parameters. DateServer provides a set of routines for date comparison, calculation and manipulation. It supports 36 different date formats, and provides routines for date validation, comparison, conversion, and determination of day of week.

TEXT:

Computer Software Corp. Prepares Legacy Systems For Year 2000

For many organizations, preparing legacy systems to correctly process multicentury dates will likely require a major expenditure of time and money over the next few years. Changes must be made to all application systems that make date field comparisons or calculations using date field values with only a two-digit representation for the year. Computer Software Corp.'s DateServer 2000 software offers a unique solution to significantly reduce the time and effort to prepare for processing in tile year 2000 and beyond.

The conventional approach to the year 2000 problem is to expand date field sizes to provide for date values with a four-digit year. This file-conversion, or "sledgehammer," approach is a costly solution for most enterprises. It requires special programs to convert all date fields within all databases. Changes to all existing input screens and files are also required. It requires coding changes to all current programs that process the new input dates and the converted databases as well as changes to all output screens and reports. Finally, this conventional approach requires the testing of all the new and changed programs, and a great deal of coordination during the actual system conversion and implementation.

No Database Conversions

In contrast, the DateServer 2000 solution eliminates the requirement to convert any files or databases. Using the familiar CALL statement interface, programmers can access DateServer 2000 routines to make date field comparisons or calculations on all date fields in their present database formats. During program execution, DateServer 2000 routines will assign the appropriate century value, based on the system's current date and DateServer 2000 installation parameters, to correctly perform requested date comparisons or calculations.

One Program At A Time

Eliminating the need for file or database conversions and the requirement to expand input or output date formats, the DateServer 2000 solution simplifies the preparation and coordination efforts required to process multicentury dates. Application programs may be changed, tested and implemented one program at a time without worry about complex database interaction with other on-line or batch programs. Rather than having to fit a major database conversion into a busy project schedule, these application program changes could be merged with other project testing.

DateServer 2000 software provides a comprehensive, efficient and

convenient set of routines for date comparison, calculation and/or manipulation. The routines support 36 date formats including character, packed and binary Gregorian and Julian dates, and day of week and relative day values for all dates in the Gregorian calendar. The routines perform: date validation; date comparison; conversion from one format to another; determination of day of week and relative day values; calculation of the difference between any two dates in number of years and/or days; and determination of the date resulting from the addition or subtraction of any number of days to or from a given date.

Even enterprises that choose the file conversion approach for some legacy systems could still use the DateServer 2000 solution for other systems.

Computer Software Corp. offers versions of the DateServer 2000 software to operate in either an IBM MVS or IBM VSE environment. A C1CS demonstration program is also included for CICS users. Priced on a tiered basis with multiple-site discounts, DateServer 2000 software is available starting at $10,000. For more information, contact Computer Software Corp., 19100 Detroit Road, Cleveland, OH 44116, (800) 908-2000, Fax (216) 333-8288, or (216) 333-9080 outside the United States.

COPYRIGHT 1994 Cardinal Business Media, Inc.

SPECIAL FEATURES: illustration; chart
COMPANY NAMES: Computer Software Corp.--Products
DESCRIPTORS: Product Information; DBMS Utility; Product Description/Specification
SIC CODES: 7372 Prepackaged software
TRADE NAMES: DateServer 2000 (DBMS utility)--Design and construction
OPERATING PLATFORM: DOS/VSE
FILE SEGMENT: CD File 275

7/9/1 (Item 1 from file: 88)
DIALOG(R)File 88:Gale Group Business A.R.T.S.
(c) 2000 The Gale Group. All rts. reserv.

02471390 SUPPLIER NUMBER: 08831360 (THIS IS THE FULL TEXT)
Databases.
Neuhaus, Trudy
PC Magazine, v9, n16, p471(4)
Sept 25, 1990
ISSN: 0888-8507 LANGUAGE: English RECORD TYPE: Fulltext
WORD COUNT: 855 LINE COUNT: 00088

TEXT:

DATE CONVERSION

This is a user-defined function that converts a date expression in DBASE IV into a more readable version. I use this simple function with all of my code; it makes any computer-generated report more personable.

```
FUNCTION cdate
 PRIVATE mdate
 PARAMETER mdate
 RETURN  (CMONTH(mdate) +  " " +;
  LTRIN(STR(DAY(mdate),2)) +;
  ", " + STR(YEAR(Mdate),4))
```

For example,

```
 mbirth = CTOD("3/10/61")
 ? CDATE(mbirth)
 * Result: March 10, 1961
```

This also can be expanded to include other information such as the day of the week.

Benjo Dans
Saginaw, Michigan

Indeed, date functions like this one are a mainstay in most programmers' DBASE toolkits. Many variations on this theme are possible. For example, to create a function that also returns the day of the week, replace the RETURN statement above with the following:

```
RETURN (CDOW(mciate) + ", " +;
  CMONTH(mdate) + " " +;
  LTRIM(STR(DAY(mdate),2)) +;
  ", " + STR(YEAR(mdate),4))
```

The resulting function works like this:

```
 mbirth = CTOD("3/10/61")
 ? CDATE(mbirth)
 * Result: Friday, March 10, 1961
```

Or, if you need a shorter version for a report with space limitations, try

```
RETURN  SUBSTR(CMONTH(mdate),1,3) +;
  " " + LTRIM(STR(DAY(mdate),2)) +;
  ", " + STR(YEAR(mdate),4))
```

which works like

```
mbirth = CTOD("3/10/61")
 ? CDATE(mbirth)
 * Result: Mar 10, 1961
```

FoxPro users can use a built-in function called MDY, which returns the date in the first format above. However, the year portion is dependent on the SET CENTURY setting. For example, MDY(mdate) returns "March 10, 61 " with SET CENTURY OFF (note that "19" is missing). In addition, CENTURY defaults to OFF.

To ensure a complete four-digit year, you may be tempted to build a UDF that locks in the century setting:

```
FUNCTION mymdy
 PARAMETERS mdate
 PRIVATE savecen, rdate
 savecen = SET("century")
 SET CENTURY ON
 rdate = mdy(mdate)
 SET CENTURY &savecen
 RETURN rdate
```

This saves and restores the current century setting while returning a

date guaranteed to have four digits. However, you'd be well advised to resist any such temptation, since Mr. Dans's much shorter CDATE function executes about 15 times faster than mymdy and accomplishes the same thing.

CONVERTING NUMBERS TO WORDS

I'm submitting a FoxBASE 2.01 program that converts a numeric amount to its English equivalent--a facility necessary in my invoicing program.

Alfredo Conde

MIA Airport, Philippines

A common application for this technique is printing checks. Mr. Conde's original version worked quite well, but only for amounts up to $999,999.99. Such a limitation won't do! The N2W procedure in Figure I has been changed so that it works with numbers up to $999,999,999,999.99 (just in case you'd like to place a deposit against the national debt). In tests against a file of 50 random numbers, it worked fine.

To see how it works, try the following example:

```
SET PROC to NWORDS
? N2W(150.99)
* Result is:
* One Hundred Fifty Dollars and
* 99/100
```

You should note that this routine works in FoxBASE +. It does not work in current releases of FoxPro (through 1.2).

LEFT-PADDING WITH ZEROS

Recently I faced a perplexing DBASE problem. My client wanted a database unloaded in SDF (System Data Format) with all numeric fields zero-padded to the left of the most significant digit. How easy, I thought. Simply define the field as character data and use the STRO function to convert any numeric data to character data. But the STR() function doesn't left-zero-pad. So I came up with a technique that combined the STR(), SUBSTR(), and STUFF() functions. The example that follows will left-zero-pad a seven-position field containing the number 257:

```
STORE STR(257,7) to x
STORE 1 to I
DO WHILE I <= 7
  IF SUBSTR(x,i,1)=' '
    STORE STUFF(x,i,1,'0') to x
  ENDIF
  STORE I+1 to I
ENDDO
```

Terry Blankenship

Brentwood, Tennessee

Numeric formatting is a more common operation required by many business programs. This technique does the job but may be shortened. It can be made to work with any length string by including the SUBSTR comparison as part of the DO WHILE loop:

```
STORE 1 to I
DO WHILE SUBSTR(x,i,1)=' '
  STORE STUFF(x,i,1,'0') to x
  STORE I+1 TO I
ENDDO
```

If your DBASE implementation supports user-defined functions, you can create one to do the converting and padding like this:

```
*
 * PROCEDURE ZNUM(number, slen)
 *
 * This function converts a number
 * to a string of length slen left
 * padded with leading zeroes.
 * Returns the string.
 *
PROCEDURE ZNUM
PARAMETERS number, slen
STORE 1 to I
STORE STR(number, slen) to string
DO WHILE SUBSTR(string,i,1)=' '
  STORE STUFF(string, i, 1, '0') to
   string
  STORE I+1 to I
```

```
ENDDO
RETURN string
```

To use ZNUM in your program, you could type

```
string = ZNUM(257, 7)
* "0000257"
```

Readers will be interested to note that Fox Software's FoxPro includes built-in functions called PADL, PADR, and PADC to left-pad, right-pad, and center-pad, respectively. If you're using FoxBASE+, here's a handy emulation of FoxPro's PADL function:

```
*
 * PROCEDURE PADL(string, length,
 *  padchar)
 *
 * This UDF left pads a string to
 * length characters padding with
 * character padchar. It is similar
 * to FoxPro's PADL in that it
 * truncates the string if the
 * string is longer than length, and
 * does not trim leading or trailing
 * blanks on the incoming string.
 *
PROCEDURE PADL
PARAMETERS string, length, padchar
IF LEN(string) > length
    STORE SUBSTR(string,1,length) TO
     string
ENDIF
DO WHILE LEN(string) < length
   STORE padchar + string TO string
ENDDO
RETURN string
```

The PADL function accepts three arguments: the string to pad, the desired length, and the character to use for padding. You could use it in your programs like this

```
string1 = PADL("257",7,"0")
 * string1 = "0000257"
 string2 = PADL("257",7,"*")
 * string2 = "****257"
```

If you're using DBASE TV or FoxPro, another alternative is the TRANSFORM function. This function lets you apply PICTURE formatting to character, logical, date, and numeric data and returns the resulting string.

The function codes for leading zeros and right justification are L and R respectively, so the following does the trick:

```
string1 = TRANSFORM(256,"@LR
  9999999")
  * string1 = "0000257"
```

LET US HEAR FROM YOU

Share your database experience, tips, and techniques and we'll pay you $50 for any submissions we print. Submissions must be made on-disk; be sure to include a printout, too. Mad your contributions to Databases, PC Magazine, One Park Avenue, New York, NY 10016, or upload them to PC MagNet (see the "By Modem" sidebar in the Utilities column for instructions.)

COPYRIGHT 1990 Ziff-Davis Publishing Company

FILE SEGMENT: CD File 275

3/9/1 (Item 1 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)
(c) 2000 The Gale Group. All rts. reserv.

01586145 SUPPLIER NUMBER: 13450166 (THIS IS THE FULL TEXT)
Julian **and Gregorian calendars: data-conversion functions for yesterday and today. (includes related articles on how Gregorian calendar was adopted, proleptic calendars) (Tutorial)**

Meyer, Peter J.G.

Dr. Dobb's Journal, v18, n3, p152(5)
March, 1993
DOCUMENT TYPE: Tutorial ISSN: 1044-789X LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 1900 LINE COUNT: 00144

ABSTRACT: Program techniques for converting dates between the Julian and Gregorian calendar systems are presented. The Julian calendar assumes a year of 365 days and six hours and exceeds the current true value by approximately 11 minutes; Pope Gregory XIII had the calendar reformed in 1582 to correct the error. A C function which converts any date within a period of eleven million years in either the Julian or Gregorian calendar into a unique 'long int' number is presented. Julian-day numbers are still used by astronomers. The C function can be used to determine the day of the week of any date and the number of days between any two dates.

TEXT:

Date-conversion functions for yesterday and today

The Western calendrical system--known as the Julian calendar and consisting of a year of 12 months and of 365 days with an extra day every fourth year--was established by Julius Caesar (following the advice of the Alexandrian astronomer Sosigenes) in 46 B.C. The extra day may not have been added consistently until A.D. 8, during the reign of Augustus. Subsequently, this calendar became widespread as a result of the expansion of the Roman Empire. The system of numbering years Anno Domini was instituted in A.D. 525 by the Roman abbot Dionysius Exiguus.

The Julian calendar assumes that the average length of a year is 365 days and six hours (since one day is added every four years). The length of the year assumed in the Julian calendar exceeds the current true value by about 11 minutes, resulting in an error of about three days every 400 years. Thus, as the centuries passed the Julian calendar became increasingly inaccurate with respect to the solar year as defined in terms of the solstices and the equinoxes. This was especially troubling to the Church because it affected the determination of the date of Easter, which by the sixteenth century was slipping gradually into summer. To resolve these problems, the calendar was reformed in 1582 on the authority of Pope Gregory XIII, and the modified calendar is called the Gregorian calendar.

In this article, I'll present a C function which converts any date within an 11-million-year period in either the Gregorian calendar or the Julian calendar into a unique long int, a number in the range of approximately - 2,000,000,000 through 2,000,000,000. A function is also given for conversion of a long int back into a date in one of the calendars. This permits conversion between dates in the Julian and Gregorian calendars and provides a basis for other date-manipulation functions. The date-conversion functions given in this article are used in a general C-function library that I developed, the Dolphin C Toolkit.

Universal Date Conversion

According to the Gregorian reform, ten days (or more exactly, dates) were omitted from the calendar. It was decreed that the day following October 4, 1582 (which was October 5, 1582 in the old calendar) would thenceforth be known as October 15, 1582. In addition, the rule for leap years was changed. In the Julian calendar, a year is a leap year if it is divisible by 4. In the Gregorian calendar, a year is a leap year if it is divisible by 4, with the added criterion that years divisible by 100 must also be divisible by 400. Thus the years 1600 and 2000 are leap years, but 1700, 1800, 1900, and 2100 are not. Finally, it was decreed that new rules for the determination of the date of Easter would be adopted.

Day Numbers

Astronomers use a system of numbering days called Julian-day numbers. The term "Julian-day number" (unlike the term "Julian calendar") does not

derive from the name of Julius Caesar. This numbering system is said to have been named after Julius, the father of its inventor. The astronomical system of Julian-day numbers should not be confused with the simpler system of the same name, which associates a date with the number of days elapsed since January first of the same year (according to which December 31, 1993 is Day 365).

The astronomical Julian-day number of the day specified by a date in either the Julian or the Gregorian calendar is the number of days elapsed from the day designated as January 1, 4713 B.C. in the Julian proleptic calendar. (See the textbox entitled, "The Proleptic Calendars" for more information.) Thus, the Julian-day number of 1/1/4712 (J) is 0. Note that 4713 B.C. is the year-4712. The Julian-day number of 10/10/1992 is 2,448,906.

There is a simple relationship between Gregorian-day numbers, as used in this method of date conversion, and Julian-day numbers. Given a Gregorian-day number, the corresponding Julian-day number is obtained by adding 2,299,161, the Julian-day number of October 15, 1582).

Listing One, page 158, contains the header file, DATECONV.H, used by the date-conversion functions presented in this article. A structure Date contains values for day, month, and year, plus a value gdn (for Gregorian-day number, as explained shortly) and a flag indicating the validity of the date. A day/month/year value is ambiguous until the particular calendar is specified. A date is completely specified using an instance of the structure together with an instance of a separate charvariable that has the value G or J.

We first need to ascertain whether a given year is a leap year (in the Julian or Gregorian calendars). Functions to do this are given in DATECONV.C (see Listing Two, page 158). A universal date-integer conversion method, as understood here, consists of two functions: The first takes a date (either Julian or Gregorian) and returns a positive or negative integer (long int); the second takes a long int and a calendrical specification (G or J) and returns a date in that calendar. It does not matter which date corresponds to day 0 as long as there is a quickly computable, one-to-one correspondence between dates in a calendar and numbers.

The method presented here converts dates into the number of days before or after October 15, 1582--the day that the Gregorian calendar came into effect. Thus, October 15, 1582 (Gregorian) corresponds to day 0; October 16, 1582 (Gregorian) to day 1; and October 14, 1582 (Gregorian) to day -1. The number corresponding to a date is thus called the "Gregorian-day number." Dates in the Julian calendar, as well as those in the Gregorian calendar, are mapped into Gregorian-day numbers. Thus, the day preceding October 15, 1582 in the Gregorian calendar is both October 4 in the Julian calendar and October 14 in the Gregorian calendar--both have Gregorian-day number -1.

The code for the function to convert a date in one of the calendars to a Gregorian-day number, and for the function to convert a Gregorian-day number to a date in a specified calendar, is given in Listing Two. The Date structure uses long int variables (signed) for the year and the Gregorian-day number. The largest integer representable as a signed long int is 2,147,483,647. Due to the method of calculation, however, the largest Gregorian-day number that can be used is 2,146,905,911. This corresponds to the dates July 11, 5,879,611 (Gregorian) and October 19, 5,879,490 (Julian). By that time, dates in the Gregorian and Julian calendars will differ by about 121 years. (This will be of no practical importance since by that time both calendars will likely have been superseded.)

To use the conversion functions, declare a structure of type Date (defined in Listing One) and pass to the functions a pointer to the structure along with a calendrical specification (G or J). If you are converting from Gregorian-day number to date, define the gdn structure variable before calling gdn[underscore]to[underscore] date(). Conversely, define the variables day, month, and year before calling date[underscore]to[underscore]gdn(). On return from the functions, extract either gdn or the date values from the structure. Before using the gdn value, it is advisable to check the validity flag, which will be TRUE if the date passed was a valid date in the specified calendar, and FALSE otherwise. For example, the attempt to convert February 29, 1900

(Gregorian) to a Gregorian-day number will produce a FALSE in the validity variable.

The lfloor() function in Listing Two is analogous to the Microsoft floating-point library floor() function, and overcomes a small problem in integer arithmetic. The date-conversion functions described earlier need a long-integer division operation such that, for all long integers a and b, a/b is the largest integer not greater than the real number a/b. MSC's division operator produces this result if a and b are positive, but not if a is negative and b is positive. The lfloor() function provides what is needed.

Finally, DATECONV.C also contains a function to convert a Gregorian-day number into a day of the week. This is independent of the particular calendar used.

Testing

No function or program can be relied upon unless it is tested thoroughly. The program DATETEST.C in Listing Three (page 159) tests the functions in Listing Two. DATETEST takes two numbers, n and m, on the command line and performs conversions for n, n+m, n-m, n+2*m, n-2*m, and so on, up to the largest integer that can be handled. It first converts the number to a date using gdn[underscore]to[underscore]date(), then converts the resulting date back to a number using date[underscore]to[underscore]gdn(). The program does this for both the Gregorian and the Julian calendars. If the number resulting from converting a number to a date and back to a number were different from the initial number, then a bug would be revealed.

If DATETEST is run with command-line parameters 0 and 1, then all dates forward and backward from October 15, 1582 (Gregorian) are tested, although the program displays the results only for every Nth conversion. (N is currently defined as 3000 in Listing Three.)

Since there are over four million input numbers that could be tested, exhaustive testing is not practical unless you can run the program on a very fast computer. It has been shown, however, that when using DATETEST 0 1, no bugs are revealed for all Gregorian-day numbers in the range -14,235,000 through 14,235,000. The corresponding dates include all dates in both calendars for all years from -37,390 to 40,555.

Derivative Calendrical Functions

Given the basic date-number conversion functions, it is not difficult to develop other calendrical functions. In fact, there are 38 date functions in the Dolphin C Toolkit, including functions to determine which of two dates is earlier, how many days separate two dates, and how many weekdays separate two dates. There is also a function to format a date in hundreds of different ways.

Conclusion

The Dolphin C Toolkit includes a demonstration program, CAL[underscore]FNS, which allows conversion between dates and Julian-day numbers. This program also provides the day of the week of any date and the number of days between two dates. CAL[underscore]FNS allows us to determine, for example, that the storming of the Bastille occurred on a Tuesday. We can also discover that on the evening of April 18, 1521, when Luther defended himself against charges of heresy before the Holy Roman Emperor, Charles V, it was a Thursday. Finally, the coronation of Charlemagne as Holy Roman Emperor in Rome on Christmas day, 800, occurred on a Friday. We may reasonably suppose that festivities continued throughout the weekend.

COPYRIGHT 1993 M&T Publishing Inc.

DESCRIPTORS: File Format Conversion Software; Calendars; Program Development Techniques; Programming Instruction; Tutorial; Time Measurement
FILE SEGMENT: CD File 275

3/9/2 (Item 2 from file: 647)
DIALOG(R)File 647:CMP Computer Fulltext
(c) 2000 CMP. All rts. reserv.

00567181 CMP ACCESSION NUMBER: IWK19900226S2654
PREPARING FOR 2000 - **On the first day of the next decade, millions of Cobol programs may be wrong**
John **Xenakis**
INFORMATIONWEEK, 1990, n 259, 19
PUBLICATION DATE: 900226
JOURNAL CODE: IWK LANGUAGE: English
RECORD TYPE: Fulltext
SECTION HEADING: TE
WORD COUNT: 675
TEXT:

Hidden within the billions of lines of code that make up software programs across the nation is a time bomb. On the first day of the next decade, the first of the next millennium, millions of computer programs written in Cobol, PL/1, and other languages, will start giving out the wrong answers.

The problem is the way in which standard programs have represented dates. The only standard way the current date is identified in a Cobol program returns the date in the six-digit form YYMMDD, with two digits for the year. Feb. 26, 1990, for example, is represented as 900226, and Jan. 1, 1991, is represented as 910101. The former is smaller than the latter, so the computer assumes that the first date is earlier than the second one. And almost all Cobol programs depend on that fact to determine the answers to questions related to such things as inventory expiration and interest computations.

But on Jan. 1, 2000-that is, 000101-those comparisons will fall apart. Programs which depend on such computations will get the wrong answers. Even standard sort utilities, which arrange records of a file in order by date, will fail. In order to work correctly, these programs and files will have to be modified to store a four-digit year , such as 19900226 or 20000101, in the format YYYYMMDD.

"About 75% of all the Cobol programs out there are going to have to be modified," says William Strapko, an analyst with market researcher International Data Corp. in Framingham, Mass. "The biggest problem is to convince management that they're going to have to dedicate a fair amount of resources in order to resolve this problem. It won't be easy."

"Can you recompile old programs and have them run correctly?" asks Carl Cargill, a standards consultant at Digital Equipment Corp. in Maynard, Mass. "That will be affected by how well the programs were written. The IS managers who hired cheap entry-level programmers got the job done cheaply, but now they're going to have to pay for it."

Of course, many programs were written carefully to start with. "In my last life, I used to work on an international banking system," says Marc Sokol, executive VP at Chicago-based Realia Inc., a vendor of Cobol compilers. "We had to keep track of maturities that expired in 20 years, so we were very aware of the problem."

Bill Schoen, an independent consultant in Clarkston, Mich., points out, however, that such prescience is rare. "I've worked as a consultant to Chrysler, Ford, and EDS Electronic Data Systems Corp.!," he says, "and I know they've done very little so far to handle this problem. I can guarantee that even many systems being written today are going to fail in 2000."

DEC's Cargill says computer and compiler vendors are going to be under pressure to help their customers. "The user will say, `I have a lot of money invested in your equipment, and I'm going to invest a lot more. What are you going to do to fix this?' The problem is that the user will have to convert his own programs."

To aid conversions, the ANSI Cobol standard has recently been modified to accommodate a new intrinsic function named Current-Date, which gives the current date with a four-digit year. Vendors are required to implement this function within two years if they wish to remain compliant.

Some IS managers are hoping for a program that will convert old Cobol programs to use the new intrinsic function automatically, but Schoen says a lot of manual work is still needed. "Even if new technology caught

98% of all the errors, the other 2% will be a disaster."

In fact, there is a precedent for this situation. In 1979, the U.S. Postal Service went from a five-digit zip code to a nine-digit zip code. "It was a horror show," says Cargill.

The difference, of course, is that you don't have to change your zip code software. The year 2000 won't offer that choice.

90/005,592

Analysis&Perspective

SOFTWARE PATENTS

In addition to the highly publicized *Dickens* patent, a number of other patents contain claims covering aspects of Y2K remediation techniques. A broad examination of potentially applicable patents is necessary to create a comprehensive strategy for dealing with potential patent infringement liability.

Patented Y2K Solutions: Have You Looked To See Which Ones You May Be Using?

BY BLANEY HARPER AND DAVID COCHRAN

Recently, the press has reported that Mr. Bruce Dickens, holder of United States patent 5,806,063 entitled "Date Formatting And Sorting For Dates Spanning The Turn Of The Century," is seeking license fees from alleged users of his patented technique for solving the Y2K problem.[1] At least some in the high tech industry have been taken aback by the fact that a technique for fixing Y2K programming problems has been patented.[1]According to Giga Research Group however, over 30 patents have issued to date on solutions for the Y2K problem.[2] And, while there is no way to tell, it is likely that other such patents are still in the pipeline at the United States Patent and Trademark Office. Furthermore, like Mr. Dickens' patent, most of these Y2K patents are owned by individuals or small entities, which are just the type of entities likely to be seeking license fees in the multibillion dollar Y2K industry.

To shed some light on the potential scope of some of these Y2K patents, this article groups the patents into three broad categories and then reviews some of the broadest claims contained within each group. As discussed below, the sum of these patents covers a wide range of potential Y2K solutions that generally fall into three groups: A) date "window" techniques, such as the *Dickens* patent; B) numerical format conversions; and C) file system techniques. While it is beyond the scope of this article to fully analyze the specifications, file histories, prior art or other applicable evidence to properly construe any of these claims, we will examine the claim language associated with these groups of Y2K solutions and provide a general overview of the total subject matter potentially covered.

Firms engaged in Y2K remediation are likely to have used a variety of different vendors who each may have used a variety of different techniques for solving the problem. Thus, such firms should evaluate their programming solutions and potential patent infringement liability against the full group of Y2K patents, not simply *Dickens*.

A. Window Techniques:

I) Dickens

In light of Mr. Dickens' well publicized licensing campaign, the place to start a review of Y2K patent claims is with his patent — United States Patent No. 5,806,063. This patent was filed on October 3, 1996 and is apparently assigned to Mr. Dickens.[3] Claim 1 recites:

> 1. A method of processing symbolic representations of dates stored in a database, comprising the steps of:
>
> providing a database with symbolic representations of dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;
>
> selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;
>
> determining a century designator C_1C_2 for each symbolic representation of a date in the database, C_1C_2 having a first

[1] As this article was going to press, the United States Patent and Trademark Office announced that the Dickens patent will undergo re-examination.

[1] Y2K Fixers Are Outraged By Patent Payment Demand, The National Law Journal, p. B9, Dec. 13, 1999.

[2] Multiple Y2K fixes could touch off legal maelstrom, Erich Luening, Staff Writer, CNET News.com, November 17, 1999, URL: http://news.cnet.com/category/0-1009-200-1451541.html.

[3] Although the assignment data on the face of the *Dickens* patent indicates that it is assigned to McDonnel Douglas, and as of this date there are no assignment records in the United States Patent and Trademark office indicating otherwise, Mr. Dickens alleges that the patent has been re-assigned to him.

Mr. Harper and Mr. Cochrdn are associates with the law firm of Jones, Day, Reavis and Pogue. They are both part of the firm's Technology Issues Practice. This article reflects the views of the authors and does not necessarily represent the views of the firm or any of its clients.

 COPYRIGHT © 2000 BY THE BUREAU OF NATIONAL AFFAIRS, INC., WASHINGTON, D.C. ECLR ISSN 1098-5190

value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B; and

reformatting the symbolic representation of the date with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2 to facilitate further processing of the dates.

The literal scope of this claim language is directed to a "method of processing symbolic representations of dates." By claiming a method, Dickens sought to avoid any structural limitations and thereby encompass as many different computer systems as possible. Moreover, the language of the claimed method includes a step that is arguably present in any date-related computer system whether or not the Y2K problem is at issue. The step of "providing" a database with a representation of dates is not specific to Y2K solutions. Hence, this limitation does not appear to restrict the claim scope in any practical way.

In essence, the Dickens claim reduces to i) "selecting a 10-decade window," ii) "determining a century designator" having a first value within a date window and a second value outside the date window (*i.e.*, all dates having a year of '00' to '49' are assigned with a '20' century designator while '50' to '99' have the '19' century designator), and iii) "reformatting" the existing dates with the century designator. The "selecting" and "determining" steps simply recite the logic embodied in the window technique. The full scope of the reformatting step, however, is not defined in the patent. While the patent does give an example of reformatting — changing YYMMDD format representations to CCYYMMDD format representations — the claim language is not literally limited to that type of reformatting.[4]

Due to the lack of explanation in the specification, the boundaries of the reformatting step (and hence the claim as a whole) must be determined by reference to other evidence such as the file history or expert testimony. Notwithstanding the uncertainty in the actual claim boundary, claim 1 at least purports to cover methods of Y2K solutions in which conventional six-character dates (YY-MM-DD) are associated with the proper century designator (CC) based on creating a window in a 100-year interval. Accordingly, this claim potentially applies to a broad swath of window solutions to the Y2K problem.

ii) Baird

The *Baird* patent (No. 5,758,346) is a variation on the windowing theme of *Dickens*. This patent was filed on January 29, 1997 (after *Dickens*) and is assigned to Electronic Data Systems Corporation of Plano, Texas. In *Baird*, a window is selected so as to facilitate date conversion. Specifically, the broadest method claim in this patent recites:

13. A method for converting representations of year, the method comprising the steps of:

receiving a value for each of a set of parameters for defining a floating window of years;

configuring the floating window of years in response to the received values;

receiving a representation of year in either a two-digit format or a four-digit format;

determining whether the represented year falls within the floating window of years;

converting the representation into the other of the two-digit format or the four-digit format if the represented year falls within the floating window of years; and

rejecting the representation if the represented year does not fall within the floating window of years.

In the *Baird* method, conventional steps of configuring a window and receiving year data in a 2 or 4-digit format are followed by a comparison of the year data to the defined window. If the year data falls within the window, that year data is converted from its existing 2 (or 4) digit form to a 4 (or 2) digit format. If the year data falls outside the defined window, it is not converted to another format. While *Baird* is distinct from *Dickens* because of, *inter alia*, *Baird's* non-conversion of dates outside the window, Y2K solutions that infringe *Dickens* may still nonetheless infringe *Baird*. In particular, converting 2-digit year dates to 4-digit year dates may meet the reformatting step of *Dickens*. Hence, a variety of Y2K solutions may be covered by both *Dickens* and *Baird*.

iii) Alter

In another example of the date window theme, the *Alter* patent (No. 5,600,836) purports to cover a more specific Y2K technique for determining the proper century for years in a 100-year window. This patent was filed on November 14, 1995 (before *Dickens* and *Baird*) and is assigned to Turn Of The Century Solution, Inc. of Wayne Pennsylvania. The broadest method claim in the *Alter* patent recites:

11. A method for processing date-dependent information using a computer system; the date-dependent information including dates in at most two centuries wherein the date-dependent information comprises a year field specified in a two-digit format, the method comprising the steps of:

converting input date data to be processed from local time to zone time and defining the input date data so converted as zone input date data;

processing the zone input date data in accordance with an existing application integrated into the computer system to generate zone output date data; and

converting the zone output date data into output date data.

The *Alter* claim language recites three steps: i) converting 2-digit dates into zone dates, ii) processing the converted input dates to generate output dates, and iii) converting the output dates to local dates. Input dates are converted to zone dates by an algorithm that, in effect, uses a selected date window to shift all dates into a single century. Once these converted input dates have been processed to generate output dates, the output dates are then shifted back. *Dickens* and *Baird* are distinct from *Alter* because, *inter alia*, they do not require processing of the date data in an existing application followed by a conversion of the processed output date data as in *Alter*. Because *Alter* requires an analysis of the interaction of application programs with date data files, rather than just an analysis of date data files as in *Dickens*and *Baird*, *Alter*covers distinct subject matter.

B. Numerical Format Conversions:

In addition to the Y2K patents implementing date windowing techniques as described above, there are a series of Y2K solutions in which the numerical format

[4] By implication, independent claim 1, which does not recite a specific type of reformatting, is broader than dependent claim 5 which does recite the specific type of reformatting given in the example.

of the date representation is modified or converted to a form that embeds century information. Basically, these techniques are intended to pack more date information into the existing 6-digit space of the conventional YYMMDD format date representation. For example, in the *Brady* patent (No. 5,758,336), a specific numerical format conversion — packed binary — is used to encode century information.

Brady was filed on May 30, 1996 and is assigned to MatriDigm Corp. of Fremont, California. Claim 1 of *Brady* recites:

> 1. A method for handling date data in a computer, said method comprising the steps of:
>
> issuing a call to a date conversion subroutine when a date field is to be processed in a program running on said computer;
>
> transferring date data from said date field to said date conversion subroutine;
>
> determining from said date data, a date format thereof by examining at least one bit position of said date data;
>
> if said date format is determined to be other than a packed binary format, converting said date data to said packed binary format, wherein at least one bit of said packed binary format indicates said packed binary format, and at least some remaining bits of said packed binary format represent a binary value indicative of a four digit year value; and
>
> performing a logical operation using said packed binary format year value.

According to the claimed first step in *Brady*, a date conversion subroutine is called whenever a date field is to be processed in an application program. Next, data from the date field are transferred to the conversion subroutine and a format of the date data is determined. If the format is something other than packed binary, the date is converted into a packed binary form. Once in a packed binary form, the application program performs logical operations on that packed binary form of the date data. In this way, the application program detects the correct century information without that century information increasing data storage space.

Other Y2K patents implementing various types of numerical format conversions include:

- ***Bieler*** (No. 5,930,506; filed September 2, 1997 and not assigned), wherein either the month characters (MM) or the day characters (DD) of a conventional date format are altered such that they indicate whether the year is before the year 2000 or after the year 2000. For example, according to *Bieler*, instead of the month numbers ranging from 1 to 12, the month characters will be altered to range between 21 and 32 for years after the year 2000. Alternatively, the day field (DD) could be altered in similar fashion such that instead of the day ranging from 1 to 31, the day range can be altered to be from, for example, 41 to 71. In this scheme, the altered numerical characters in the day field (DD) indicate to the system that the year field (YY) is a date beyond the year 2000.

- ***Beam*** (No. 5,950,197; filed May 7, 1997 and not assigned), wherein, in addition to the symbols 0 through 9 for the year code (YY), other non-numeric symbols can be used to represent values greater than nine. For example, a "+" can be used to represent the number "10", or a "=" sign could be used to represent the number "11." In this manner, for example, the year code (YY) symbol "+0" would represent the year 2000 since + is a 10 and the addition of the 0 would equate to 100 years from the year 1900. Or, the symbol "+" could represent the number 12, in which case the date code for the year "+0" would be 120 years from the year 1900, and thus would represent the year 2020.

- ***Nicholas*** (No. 5,956,510; filed September 10, 1996 and assigned to Unisys Corp.), which recognizes that the year fields of a conventional date format include two 8-bit characters (YY) and that an 8-bit binary character can encode scalar values from 0 to 255. In *Nicholas*, dates that are beyond the year 2000 are encoded into the existing space used for the eight bit values by converting the year characters (YY) into their scalar representation.

- ***Wolfe*** (No. 5,970,247; filed June 17, 1996 and not assigned) wherein the conventional 6-character date format is extended beyond December 31, 1999 by encoding dates with the number 99 for the year field (YY), but coding the month and the day fields (MM-DD) with values beyond 12 and 31, respectively (*e.g.*, MMDD = "1232" or greater). This patent also includes using an alpha-numeric character in one of the month and day fields.

- ***Adamchick*** (No. 5,761,668; filed March 8, 1996 and not assigned), wherein the conventional date representation (MM-DD-YY) is converted into an alternative format in which there is a single character that is indicative of the century (C), two characters that are indicative of the year (YY) and three characters that indicate the day within the year (DDD). Thus, in this new format (C-YY-DDD), there are no characters that represent the month (MM).

- ***Gregovich*** (No. 5,797,117; filed February 7, 1998 and not assigned), where month values (MM) greater than 12 are used for the 84 months beyond the year 2000 (*i.e.*, from 13-96), thus extending the computer system's operation for a period of seven years beyond the year 2000.

- ***Mao*** (No. 5,668,989; filed September 18, 1996 and not assigned), wherein a hybrid radix 2-digit year number is used to encode years beyond the year 2000. In the hybrid radix scheme, the tens digit of the 2-digit year is represented as a hexadecimal value, *i.e.*, from 0 to 15, and the ones digit is represented as a decimal value as is traditional. By representing the year number using this radix 2-digit format, the two-digit date field represents dates between 1900 and 2059.

ARTICLE SUBMISSIONS

The editors of the *Electronic Commerce & Law Report* invite attorneys and academics to submit for publication articles addressing legal issues arising from the convergence of electronic information technologies. Good candidates for publication are articles on data protection, privacy, electronic commerce, intellectual property rights in cyberspace, computer crime, and the application of constitutional principles to emerging communications media. Articles with appeal to an international audience are most desired.

Articles currently published on the World Wide Web will be considered for hyperlinked listing in the Electronic Resources section of the Report.

Submissions should be directed to the Managing Editor, *Electronic Commerce & Law Report*, 1231 25th Street, N.W., Washington, D.C. 20037; telephone (202) 785–6883, fax (202) 728–5203, or e-mail to totoole@bna.com.

C. File Editing:

In addition to those Y2K patents using date windows or numerical format conversions, there is another group of Y2K patents that attempts to capture a broad cross section of Y2K solutions without actually specifying any particular technique for modifying a date. *Shaughnessy* (No. 5,630,118; filed November 21, 1994 and assigned to 2000 Inc.) is an example of such a patent. Here, the broadest claim recites:

> 1. A method of modifying a computer system comprising the steps of:
>
> storing at least one subroutine in a memory accessible to the computer system;
>
> modifying an application program of said computer system to include a call to said at least one subroutine, said call operative to pass at least one data field representative of at least two dates to said at least one subroutine,
>
> wherein said at least one subroutine determines which of said at least two dates corresponds to said date field according to a predetermined criteria, performs a date operation on said date field, and returns a parameter to said application program for use in further operations.

This method claim is directed to the steps associated with physically creating the routines that are used in a Y2K solution. The claim is not directed to any specific technique that solves the Y2K problem. Rather, the claim language recites i) storing a subroutine, ii) modifying an application program to include a call to the subroutine where the call passes a data field representing at least two dates, and iii) where the subroutine determines one of the two dates, performing a date operation and returning a parameter for use in further operations. None of these steps defines a specific calculation or structure for interpreting dates. However, all of these steps may be performed when operating a computer system that implements an actual Y2K solution.

The *Nolan* patent (No. 5,897,633; filed June 20, 1997 and assigned to Century Technology Services, Inc.) is directed to another type of file editing technique. This patent claims generating a supplementary file containing only the dates necessary to be changed and then operating on that supplementary file. Specifically, claim 1 recites:

> 1. A computer-implemented method of processing a legacy computer program that utilizes a database which includes abbreviated representations of dates that are accessed by said legacy computer program, said method being adapted to process said legacy computer program to attain year and comprising:
>
> examining the database to locate and identify locations of abbreviated date representations;
>
> based on results of said examining, creating an auxiliary PALM file that includes expanded representations of dates and cross-references to corresponding ones of said abbreviated representations of dates of said database wherein, in the case of ambiguity, the determination of century indications of said PALM file is predicated on given rules for expansion; and
>
> modifying certain instructions of said legacy program that reference abbreviated representations of dates in said database to effect reference to corresponding expanded representations of dates in said PALM file in accordance with said cross-references.

In this claim, a subject database is examined for abbreviated (*i.e.*, "2-digit") date codes and a supplementary PALM file is created based on this examination. The PALM file contains expanded representations (of no particularly recited format) of the 2-digit date codes. A program that references the abbreviated 2-digit date codes is then modified to use the data in the PALM file rather than the two digit representation. The *Nolan* patent notes that, "although fully automated procedures are preferred, the extent of performance of other steps analytically may also vary." Purportedly then, these claims may extend to a variety of Y2K solutions in which full date representations are aggregated in a supplemental file regardless of whether any underlying legacy programs are automatically modified based on the dates in the supplemental file.

The *Burgess* patent (No. 5,808,889; filed June 28, 1996 and assigned to Data Integrity Inc.) is directed to still another file editing technique. Rather than attempting to provide a general Y2K solution, *Burgess* focuses on a specific subtraction operation performed by an application program. Specifically, the broadest method claim recites:

> 23. A method for correcting computer code including subtraction operations involving two digit year dates, the method comprising:
>
> computerized searching for subtraction operations in computer code involving a plurality of a two digit quantities representing year dates;
>
> executing the subtraction operation on a computer to obtain a difference value between said two quantities; and
>
> adding, to all negative difference values, a plurality of two digit numbers whose sum is 100.

According to this claim, all subtraction operations involving two dates in 2-digit format are identified. When these subtraction operations have been carried out, the output is examined to determine whether any of the resulting output numbers are negative. If so, a value of 100 is added to each such negative output value to create the correct 2-digit representation of the subtraction operation. The breadth of this patent claim lies in the fact that i) subtraction operations are likely to exist in virtually all date related programs and ii) this solution is a simple one that may well be widely employed in various applications.

Other examples of patents that claim file editing techniques for solving Y2K programming problems include:

- ***Hart*** (No. 5,838,979; filed October 31, 1995 and assigned to Peritus Software Services, Inc.), which claims:

> 7. A process for computer system data migration comprising the steps of:
>
> (a) modifying a computer program unit and data exclusively associated therewith of manageable size;
>
> (b) creating a wrapper routine for such modified unit to access unmodified data;
>
> (c) repeating steps (a) and (b), also modifying data exclusively associated with modified computer program units until all computer program units are modified.

- ***Fridman*** (No. 5,926,814; filed September 22, 1997 and assigned to Consist International), which claims:

> 1. A method of selectively modifying date fields of a target application program, comprising the steps of:
>
> identifying field source code representing each date field in said target application program by analyzing source code of said target application program;

analyzing each of said field source code to determine the number of year-digits available in each said date fields;

modifying the analyzed field source code representing said date fields capable of accepting only two-digit years to modified source code representing date fields capable of accepting three-digit years; and

replacing said analyzed field source code representing the two-digit year date fields with corresponding modified source code.

■ ***Brown*** (No. 5,852,824; filed May 22, 1997 and not assigned), which claims:

4. A method for processing year-date data in a computer system, the computer system having a computer system clock, the method comprising the steps of:

setting the computer system clock to an offset time, the offset time being a time other than the actual time;

converting the year-date data representations contained in a database file so that all year-date data representations are represented by two-digit year-date data representations wherein each of the converted two-digit year-date data representations can include both positive and negative numbers which represent a 199-year span including dates from up to three centuries; and

processing the converted two-digit year-date data representations using an application program without altering the application program.

■ ***Masiello*** (No. 5,765,145; filed July 28, 1997 and assigned to B. Edward Schlesinger Jr., Michael Colton, Paul Pross and William Polkingham), which claims:

1. A method for operating a computer system and determining the correct century date for computer date fields having two years digits in the date field and with an indication for the day of the week for the correct century date including the steps of:

a) selecting a Gregorian calendar data base including a date span for the days of the week and date, including correct century date covering a continuous time period up to 400 years;

b) putting into a program file all the calendar dates from the Gregorian calendar data base of the days of the week and dates for a date span covering a continuous time period for at least portions of two consecutive centuries to obtain a date field criteria from said 400 years;

c) running a computer search to determine a single unique day of the week value for a correct century date within said 400 years having only two years digits in said date field valid for the date field criteria;<

d) including in the computer system performance of date field operations the correct determined century date valid for the date field criteria having only two years digits in said date field;

e) whereby the computer system performance of date field operations will indicate and process the correct century date for the date span selected covering the continuous time period for at least said portions of said two consecutive centuries.

■ ***Roth*** (No. 5,878,422 filed April 9, 1997 and assigned to Viasoft, Inc.), which claims:

19. A method of virtualizing an arbitrary item of data, having a first format, contained in a data source, so as to be presented in a second format, the method utilizing a data conversion definition, the data conversion definition containing indicia of record types in the data source, fields in the record type used to identify the record type and the relevant values thereof, and data fields and the corresponding format thereof, the data fields corresponding to the data to be virtualized, the method comprising the steps of:

A) generating a conversion routine for mapping the first format data fields within the records of the data source to the second format in accordance with the data conversion definition;

B) inserting code into the program adapted to invoke a data adapter hook routine;

C) using the data adapter hook routine to intercept read requests from the program;

D) substituting, utilizing the conversion routine, in a data item read from the data source, data having the second format for data having the first format;

E) using the data adapter hook routine to intercept write requests from the program;

F) substituting, utilizing the conversion routine, in a data item to be written to the data source, data having the first format for data having the second format.

D. CONCLUSION:

A review of these patents illustrates that there are a number of claims covering different aspects of multiple techniques for solving the Y2K problem. These claims cover windowing techniques, numerical format conversions, and even file editing techniques that do not necessarily recite any particular algorithm for making a Y2K date conversion. It is clear that *Dickens* is not the only patent that firms implementing Y2K remediation efforts should examine. Rather, firms evaluating potential patent infringement liability should at least examine the full scope of the Y2K patents listed above. Only by looking at all the Y2K patents can firms create a comprehensive strategy for dealing with potential infringement liability.

 COPYRIGHT © 2000 BY THE BUREAU OF NATIONAL AFFAIRS, INC., WASHINGTON, D.C.

open access" as part of a franchise license renewal. Previous cable Internet-access requirements had been imposed on license transfers flowing from AT&T's merger with Tele-Communications Inc. and planned merger with MediaOne Group.

"Today's vote in Pittsburgh was a win for consumers," said Rich Bond, co-director of openNet in a statement. "AT&T was forced to accept a self-executing, nondiscriminatory open access provision that guarantees that if AT&T voluntarily or involuntarily signs an open access agreement with any other governmental entity or becomes subject to an open access requirement anywhere else in the United States, it must offer the same terms to Internet service providers operating in Pittsburgh."

OpenNet and others are concerned that AT&T and the rest of the cable industry give preferential treatment to their affiliated Internet service providers (ISPs), Excite@Home and Road Runner.

Uniqueness, Deviation Cited. OpenNet said the Pittsburgh provision could be triggered by a ruling in favor of "open access" by the U.S. Court of Appeals for the Ninth Circuit— which is reviewing Portland, Ore.'s, license-transfer requirement—as well as by other recent decisions in Culver City, Cal, Henrico Va., Broward County, Fla., and in several Massachusetts decisions.

OpenNet said the Pittsburgh decision was "unique" because it was the first time AT&T had voluntarily agreed to a self-executing arrangement. The group said the move also deviates from AT&T's insistence that it would not provide high-speed cable customers direct access to unaffiliated ISPs until mid-2002, when its exclusive contract with Excite@Home expires.

AT&T 'Thrilled.' AT&T Cable Services Regional Vice President James M. Mazur said in a statement, "We are thrilled with the agreement, which will allow us to begin creating an advanced cable network for the City of Pittsburgh and its residents. We commend the City negotiators, who worked hard to hammer out the best possible agreement for the City. Their knowledge and commitment were outstanding. I am particularly grateful for the leadership of Councilman Dan Cohen, who provided insight and advice to the City's negotiating team as well as managing the issue in Council."

According to AT&T, Cohen said, "As a result of this long and arduous process, the City of Pittsburgh will have an asset vital to its growth in the next century, and its residents will enjoy the benefits of state-of-the-art telecommunications technology. We are pleased with the agreement and look forward to AT&Ts partnership relationship with the City of Pittsburgh going forward."

By David Kaut

Software Patents

Patent and Trademark Office Orders Reexamination of Y2K Fix Software Patent

Commissioner of Patents and Trademarks Q. Todd Dickinson Dec. 21 ordered reexamination of a computer software patent aimed at addressing the so-called "Y2K problem," according to a press statement issued by the Patent and Trademark Office.

The patent (No. 5,896,063), entitled "Date Form[illegible] ting and Sorting for Dates Spanning the Turn of t[illegible] Century," was granted to Bruce Dickens of Irvine, C[illegible] lif., on Sept. 8, 1998. It involves a method of extendi[illegible] computer processing of dates into the next century.

Dickinson ordered the reexamination after discove[illegible] of "prior art" information that was not considere[illegible] when the patent was originally examined and grante[illegible] The PTO will consider the claims in the patent to dete[illegible] mine whether, in the context of the newly discovere[illegible] prior art, the invention meets the novelty and no[illegible] obviousness criteria for patentability.

The PTO commissioner has authority under 3[illegible] U.S.C. Section 304 to order a reexamination upon [illegible] finding that "a substantial new question of patentabilit[illegible] affecting any claim of a patent is raised." Commissioner-ordered reexaminations are discretionary and rare, according to the PTO. In the past, the agency stated, such reexaminations have been initiated where significant concern about the patentability of the claimed subject matter has been expressed by a substantial segment of the industry.

The patent and the reexamination files are available for public inspection at the PTO's Public Search Room in Arlington, Virginia between the hours of 9 a.m. and 5 p.m., Monday through Friday.

In Brief

ACEC Posts Transcripts of San Francisco Meetings

The Advisory Commission on Electronic Commerce Dec. 23 posted transcripts from its San Francisco meeting on the commission's World Wide Web site.

The commission met in San Francisco Dec. 14-15 and discussed various issues, including international taxation issues (4 ECLR 1171, 12/22/99).

Transcripts of the advisory commission's San Francisco meeting are available on the ACEC's Internet site at http://www.ecommercecommission.org/sanFran/tr1214.htm.

CORRECTION

A report at 4 ECLR 1171, 12/22/99, on a meeting of the Advisory Commission on Electronic Commerce in San Francisco should have said that the telephone excise tax was imposed during the Spanish-American War. In addition, Grover Norquist, president of Americans for Tax Reform and a commission member, should not have been listed as a supporter of the National Governors' Association position.

COPYRIGHT © 2000 BY THE BUREAU OF NATIONAL AFFAIRS, INC., WASHINGTON, D.C. ECLR ISSN [illegible]

Lead Report

Year 2000

With No Y2K Disasters, Lawyers' Thoughts Turn to Recovering Vast Remediation Costs

The long-awaited dawn of the Year 2000 has come and gone, and the nation's trial lawyers appear not to have been made noticeably richer for the experience. Few computers suffered date-related errors as the world passed into the next millennium, suggesting—for now—that the predicted billions of dollars in Y2K bug litigation may never materialize.

However, corporate America has spent many billions of dollars, an unprecedented information technology expense, in order to make its computer systems ready for 2000. This bill is one that many parties may attempt to shift to third parties, such as corporate officers and insurance companies, during the coming months.

While there was agreement among lawyers contacted by BNA that insurance coverage lawsuits aimed at cost recovery are the most likely type of Y2K lawsuit that will be filed, the experts also said that latent software problems, shareholder derivative suits for overspending on remediation, and suits over patents issued for Y2K remediation methods may also be a fruitful source of litigation in the coming months—even if, as now appears to be the case, the Y2K bug has been eradicated.

Litigation over possible violations of state and federal securities laws requiring disclosure of estimated remediation costs are also on the horizon.

Remediation Cost Recovery Front and Center. It is still too early to tell whether all systems are operating without Y2K problems, according to David Nadler, a partner with Dickstein Shapiro Morin & Oshinsky in Washington. He told BNA Jan. 3 that Y2K-related problems may not surface until the end of January or the end of the first quarter in 2000. But since systems seem to be functioning smoothly, Nadler said that the insurance coverage issues related to Y2K will likely move to front center stage.

Most property insurance policies contain what is known in the industry as a "sue and labor" clause, a provision that gives the insured party a right to reimbursement of money spent to avoid imminent harm to covered property. No court has yet ruled that Y2K remediation expenses can be recouped from insurers under the "sue and labor" clause, but there are a half-dozen or so pending lawsuits raising this issue. See, for example, *GTE Corp. v. Allendale Mutual Insurance*, D. N.J., No. 99-CV-2877 (filed 6/14/99); *Port of Seattle v. Lexington Insurance Co.*, Wash. Super.Ct., King Cty., No. 99-2-26938-1SEA (filed 11/16/99).

There is no reason to believe that "sue and labor" clause litigation, a likely growth area, will be slowed by the non-events of Jan. 1, 2000. Because of the high stakes involved, the numerous, novel insurance coverage issues raised by these cases will likely be hotly contested during the coming year.

Paul Gupta, a partner and the director of the technology law group at Sullivan & Worcester in Boston, predicted an increase in the number of suits filed against insurers for recovery of money spent on Y2K remediation. The amount spent on remediation is in the billions of dollars, he said.

Since focusing on readiness is now a thing of the past, clients are now ready to focus on recouping costs, according to Richard I. Werder, a partner with Jones, Day, Reavis & Pogue in Cleveland. The main theory used so far in insurance claims is an obligation to reimburse policyholders under the sue and labor clause. Under a sue and labor clause theory, policyholders saved their insurers significant amounts of money by remediating before the date change, and are therefore entitled to reimbursement from the insurance companies for the money spent to avert business disasters. Werder told BNA that the sue and labor clause theory may be questionable. "Whether it provides a real basis for recovery remains to be seen," he said.

Charles Kerr, a partner with Morrison & Foerster in New York, says that sue and labor cases are "a long shot at best." There are two main obstacles that Kerr sees. First, policyholders likely have significant notice problems. Most sue and labor clauses have a short notice period, usually 90 days. Spending money to remediate is the triggering event for notice, according to Kerr.

Secondly, even if policyholders can get past that hurdle, they must establish that a Y2K failure would have been a covered event under a policy. Kerr says that there are only a few decisions on sue and labor clauses reported since 1850, and that those decisions lend little support to Y2K plaintiffs' claims.

No Clean Y2K Bill of Health Yet. It will be several months before there is a clean bill of Y2K health for most businesses, according to Werder. Both Werder and Nadler agreed that supply chain interruption could still occur, and could result in breach of contract and warranty suits. Nadler noted that with the use of just-in-time inventory systems in some manufacturing industries, supply chain interruptions can quickly have a damaging ripple effect.

Another software-related Y2K problem that may not have surfaced yet are system soft crashes, according to Sullivan & Worcester's Gupta.

Soft crashes occur when data becomes corrupted in a system. Though the computer system continues to operate, it can, for example, generate faulty billings. The initial data corruption may be hard to detect without careful scrutiny. Gupta used credit card billing as one example of how a soft crash could occur. There only needs to be a small error in part of a billing process to create a significant data corruption problem downstream. If incorrect numbers are generated by part of a

COPYRIGHT © 2000 BY THE BUREAU OF NATIONAL AFFAIRS, INC., WASHINGTON, D.C. ECLR ISSN 1098-5190

system, the data is often copied and used subsequently throughout a system.

Soft crashes could yet produce Y2K litigation, according to Gupta. A system that experienced a soft crash might produce an erroneous $50 overcharge to a large number of customers, Gupta theorized. While it is unlikely that a customer would sue over such a small amount, there is collectively much money at stake, making those kinds of soft crashes the target of class-action lawsuits, he said. The most likely candidates for those kinds of suits are customers who did not notice small billing discrepancies, according to Nadler.

Another potential area for lawsuits is a shareholder derivative suit alleging unlawful corporate overspending on Y2K preparedness, according to sources consulted by BNA. However, none of the sources thought that such suits were particularly viable. Werder pointed out that there was a reasonable belief in the information technology community that the expenditures needed to be made. Such reasonable belief would likely provide a good defense, based on the business judgment rule, Werder said.

Remediation Methods May Infringe. There could also be Y2K patent infringement suits filed. A number of patents have issued with claims directed toward Y2K remediation methods. One patent, for a windowing method, was issued to Bruce Dickens, and has received widespread attention. According to Harris Miller, president of the Information Technology Association of America, Dickens has sent numerous letters to companies that used windowing technology to remediate, demanding a licensing fee, and threatening a higher licensing fee after the date change.

Windowing technology makes an assumption in a date filed that if a number is less than, e.g., 50, that a "20" should be placed in front of the date. Conversely, if the number in the date field is greater than 50, the software is instructed to assume a "19" should be placed in front of the date.

The Patent and Trademark Office announced Dec. 21 that the Dickens patent will be re-examined. According to Miller, patent suits over Y2K remediation methods could be filed, depending upon the outcome of the re-exam. ITAA has been active in gathering prior art in hopes of invalidating the Dickens patent, Miller said.

The Avalanche That Wasn't. Although an avalanche of litigation doesn't seem likely, it seems highly likely that Y2K legal practices will be forced to change. However, Y2K practices won't be disbanding tomorrow, Nadler said. Even before the date change, he said, it was clear that preparedness was high, and so the likelihood of massive litigation over Y2K disruptions appeared small. Also, the passage of the Y2K Act diminished the prospects of numerous lawsuits being filed, Nadler said.

Gupta predicted that Y2K practices will be busy for the short term. He forsees an increase in Y2K lawsuits starting at the end of the first quarter of 2000, and peaking in mid-year. After that, Y2K practices will likely transform into more generalized practices covering products liability for technology, Gupta opined.

By Jennifer L. Alvey

G.R. HELM, INC.

Y2K White Paper

Home
Overview
Year 2000
Feedback
Links

Year 2000
Approach & Compliance
System Methodologies

Presented By:

G.R. Helm Information Services, Incorporated
4993 Golden Foothill Parkway, Suite 1
El Dorado Hills, California 95762-9642
(916) 933-9669

Entire contents © 1997 by G.R. Helm, Inc. All rights reserved. Reproduction of this document in any form without prior written permission is forbidden. The information contained herein ha been obtained from sources believed to be reliable. G.R. Helm, Inc., disclaims all warranties as to the accuracy, completeness or adequacy of such information. G.R. Helm, Inc., shall have no liability for errors, omissions or inadequacies in the information contained herein or for interpretations thereof. The reader assumes sole responsibility for the selection of these materials to achieve its intended results. The opinions expressed herein are subject to change without notice.

Any referenced trademarks included in this document are the property of their respective owners.

TABLE OF CONTENTS

How Could Something Like This Happen:
The Problem
The Solutions
The Process:
Process Overview Awareness
Business Exposure
Approach Selection
Software Inventory
System Impact Analysis & Cost Estimate
Prioritize System Applications Schedule
Establish Change Management Procedures
Establish Application Group Change Requirements
Define Establish Overall Test Plan
Software Modification
Test Verify Validate
Implement Support
Pilot Project
Client Considerations
Client Systems Client Assumptions & Understandings
Approach Recommendations
Overview

Primary Considerations
Secondary Considerations
Recommendation
Timeline Windowing
Method Overview
Pivot Point Windowing
Floating Windows Conclusion

Appendix A - Terminology Definitions
Appendix B - Tools

How Could Something Like This Happen?

The Problem

The origin of the Year 2000 problem stems from a long-standing practice amongst softw developers. In the early days of computing, the cost of storage space was at a premium. it was a common practice to minimize the amount of data that was physically stored in a system. This included the omission of the century from dates when storing them to a ma medium. It was readily assumed that this would not cause any computing problems since year 2000 was decades away; 2) the software would long be replaced prior to the turn of century; 3) a 'silver bullet' would be developed to resolve any date related problems; and developer would no longer be responsible for the code.

For many years, there were no problems with the date data storage approach taken by th developers. Today, in many industries, the problem remains unrealized. However, as the 2000 draws closer, more and more companies are recognizing that a problem does exist must be addressed prior to the turn of the century or they will face grave consequences.

The basis of the problem revolves around the omission of the century when manipulatin Since dates have been stored using a 'MMDDYY' or 'YYMMDD' format, the 'YY' porti continued to increment to a larger number, as the future has become the past. However, January 1, 2000, this will no longer be the case since the 'YY' portion of the date will be Suddenly all past dates become future dates since the year '00' is numerically less than a year stored in this 'YY' format. At a minimum, reports sorted by date will be sequenced incorrectly, with the most recent date always being the last date reported for 1999. At w systems will fail since the software is unable to comprehend how the current date is less previous date! Other problems will occur prior to 01/01/2000. Many software applicatio expiration dates for one reason or another (licenses, memberships, loan terms, etc.). The of these applications have been designed not to accept a date that precedes the start date period which is ended by the expiration date. Therefore, any period that is initiated in th and expires in the 2000's will be subject to rejection, since the century omission causes a failure.

It is critical that the resolution of this problem be addressed immediately. Many industry applications have remained unaffected by the problems this situation presents. Others, li mortgage banking applications, have had to deal with this since the 1970's or earlier and long since brought the application systems into year 2000 compliance. However, the ma those not yet affected will be in the next two years. Moreover, since the solution to this p involves evaluating and possibly making a change to every piece of software, a monume undertaking will be required to correct this situation. Therefore corrective action must b immediately as time is running out and this is one software project that must be complet unmovable due date!

Return to Top

The Solutions

There are basically four types of approaches that can be used to solve the year 2000 pro short of actual application replacement. The majority of these approach methods will im much of the same software applications in one way or another and with varying degrees

The most straightforward solution is known as the "date expansion" approach. This meth solving the year 2000 problem involves expanding the date field to include the century d Fields used to store dates in the format YYMMDD would be increased in size, if require store the date in a CCYYMMDD format. This entails making a change to the data struct the dates reside in, which in turn, will require changes to the software to: 1) accept the n when retrieving it from the its source, and 2) recognize the new format and manipulate it when required. (For example, when comparing time period start and end dates.)

The second solution is known as the "timeline windowing" approach. This method of res the year 2000 problem entails establishing a range of two digit years which defines a win dates. Any year processed which resides inside this window would be considered a futur belongs in the higher of two centuries. Anything outside the window is then a past date o in the lower of two centuries. The simplest way to define a window is to select a number the "pivot point" (e.g. 50). Any year less than this number is implied to belong to the hig century (e.g. - 20 if 10) and any year equal to or greater than this number is implied to b lower century (e.g. - 19 if 87). No data structure changes are required since no modificat actually being made to the size of the field in which the dates are stored. However, apply solution involves making changes to all programs that manipulate dates to include this c determination logic.

A third solution is known as the "date encryption" approach. This approach encrypts the before storage and decrypts it upon retrieval. In essence, the data takes on a new form d storage. This approach includes compression, re-sequencing, and bit coding. Compressi includes a storage format change (e.g., changing character to packed numeric) or via sto dates in character-coded formats. Re-sequencing includes a change to the meaning of the stored in the date fields (e.g., convert to DDDDDD format where DDDDDD represents number of days since some defined date starting point, such as 1900).

A fourth solution is known as the "date encapsulation" approach. This approach encapsu date data from the programs or the entire system and visa versa. This is also referred to a shifting" and involves moving dates backwards some multiple of 28 years. (28 years sin leap year and date to the day of the week relationships are the same at these intervals.) these dates are output from a program, the multiple of 28 is added back to the date to ob actual date. The opposite occurs when dates are input to a program. Since the dates mani internally by the programs never cross century boundaries, no date logic changes or stor formats are required. However, user and system interface will require changes to perfor shift.

Each approach represents a viable method of resolving the problems introduced by the y Furthermore, it may be advantageous to use more than one method in the same system o application to obtain year 2000 compliance. However, since each business has its own e policies, procedures and date usage's, a comprehensive application evaluation is require determine which method(s) provides the overall best long term solution in the time allow complete the compliance effort. This evaluation must consider business practices, critica application processes, external and internal software interfaces, event horizons and term software and hardware constraints and capabilities, and software development standards. Additionally, budget limitations, resource availability, time constraints, change manage issues, and overall cost-benefits should be considered before a final selection is made. Fi costs are too excessive, completely replacing the system may represent the most viable s becoming year 2000 compliant.

Return to Top

The Process

Process Overview

Regardless of which solution approach is used, with the possible exception of the date encapsulation approach (at a system level), several tasks must be completed to ensure th implementation of this project. Distinct project phases and steps must occur for proper management and completion of this project. This chapter defines and describes each of t steps.

Return to Top

Awareness and Business Exposure

The first and foremost step is awareness. The correction of this problem is a monumenta with minimal cost-benefit returns - except that business functions continue uninterrupted next millennium. Normal responses will be that this is simply not that big of an issue, all involved is the expansion of some dates. Contrary to this notion, this is a development p requiring formal management approval. Unlike other software development projects, wh normally impact only a small portion of an organization's entire software portfolio, this could potentially impact as much as 90% or more of this software. Formal project mana and development methodologies are going to be required to complete this effort within t remaining. Delivery of large software development projects on schedule is an infrequent Many project deadlines have been, and are extended, for a variety of reasons without ma operational impacts. This is one project with a set deadline that cannot be changed, as ti be stopped and unfortunately, the consequences of not completing this project on time c substantial!

This is not simply a technical problem, but also a business problem. In today's competiti driven business environment, many practices have been established based on the timing Retailers and distributors retain just enough inventory to supply their customers based o historical buying habits. Manufacturers will order parts based on predefined assembly sc Safety and maintenance measures require regularly scheduled servicing of a multitude of and machinery or these devices can shut down. Moreover, the overall timing of most bus events are based on predetermined scheduling practices. Because of these business relati and interdependencies, the interruption of one scheduled could have grave financial con on many other businesses. Product shortages could stop an assembly line, overdue maint checks could shut down a telephone system halting communications, overdue safety che ground an airline or close an airport, interrupting air transportation and improper date comparisons could temporarily close some financial institutions. Due to the reliance of c business operations upon the scheduling of events, and the maintenance of these schedul computers, management must be made aware of these possible disruptions and will need prepare alternative action plans should they occur. Additionally, business must take an a in verifying that their supplies and vendors are year 2000 compliant in order to prevent b interruptions.

Return to Top

Approach Selection

As indicated in the beginning of this document, there are basically four types of approac can be used to solve the year 2000 problem, short of actual replacement. Two of these m are most commonly used to resolve this problem: expansion and windowing. The follow

notes the advantages and disadvantages of these two approaches:

	Expansion	Windowing
Advantages	• No century misinterpretation. • No special processing is required for date keys or sorts. • Duplicate keys are eliminated should dates span multiple centuries. • Minimal CPU impact. • Consistent and standard solution. • No ambiguous meaning to raw data.	• No data storage structural changes required Programs accessing dates, but not performi date logic tests or date sorts remain unaffec Application centric implementation. Progra level compliance also available. Expansion still occur, if desired or when required for external interfaces. Minimizes change management efforts since individual progra implementation can occur. Minimal ambig meaning to raw data. Reduced overall conversion effort since fewer programs are changed. No bridging is required between converted & unconverted applications. Minimizes triage.
	• Structural changes required to all date storage fields, impacting all accessing software units (programs, JCL, command files and UDC's). • Programming changes are required to all programs accessing dates, regardless of usage. • Change management and concurrent development issues becomes critical. • Temporary bridges and filters are required to interface converted and unconverted application systems. • Data centric implementation. • CPU impacted to process bridges. • Data storage requirements increase. • Missing source code for programs accessing dates must be rewritten or permanently bridged. • Additional changes may be required due to compiler upgrades or utilities no longer supported but used in older programs.	Additional logic must be added to programs that perf date logic tests or date sorts. Duplicate dates become inevitable when dates fully span multiple centuries. Windowing inconsistencies may be introduced. CPU impacted to process additional windowing logic.

The "date encryption" and "date encapsulation" approaches have been excluded from th comparison since these approaches are not recommended. Depending on the encryption used, many of the same disadvantages of one or the other two approaches are introduced the same number of advantages or realized benefits. Furthermore, raw data can become ambiguous to technical support personnel. Date encapsulation does have a primary adva that, if implemented properly, no actual coding or file expansion changes should be requ However, the proper implementation of this approach is very difficult. Additionally, "understandable" ambiguity is also introduced into the raw data, which an unwitting sup person may actually attempt to correct.

The final approach selection to be used is dependent on several factors. These include b practices, critical application processes, external and internal software interfaces, event

and time periods, software and hardware constraints and capabilities, software develop standards, and additionally; budget limitations, resource availability, time limitations, ch management issues, and overall cost-benefits. Many of these factors cannot be fully asse after the impact analysis and/or the completion of a pilot project. There have been sever organizations that have switched approaches after starting down the path of implementin realizing that for their systems and business practices, the other approach was more appr This can be both costly and time consuming. Therefore, careful assessment of the impac results should be reviewed and a pilot project implemented.

Return to Top

Software Inventory

The undertaking of this project involves potentially making a change to 90% or more of programs in an organization's software portfolio, depending on the approach selected. A minimum, every application unit and date retention area must be examined and reviewe determine the impact of a date processing change or format expansion. To be able to pe analysis properly, a complete software inventory is required. All object code and matchi code must be located and missing code noted. All JCL must also be located. Unused ap units should be discarded. Software should be grouped by application, date storage loca usages, coding language, complexity, lines of code and department ownership may also Therefore, each software unit and file can be tracked during the modification cycle. Sinc extensive inventory is required, tools should be acquired to assist in the completion of t (See appendix C.)

Return to Top

System Impact Analysis & Cost Estimate

The year 2000 compliance endeavor is a significant development project undertaking. F most part, it is not a difficult task to accomplish on a software unit level. However, sinc every program is affected, the significance of this change is enormous. Every applicatio program must be analyzed to determine the impact of a date processing change or forma expansion.

All application programs, JCL and command files, CopyLib's and data dictionaries, syst utility library functions and any special software tools created to assist software develop personnel must be reviewed. Some measure of difficulty to modify these software units also be determined. This can be done by program function, application classification, co language or by using some other measurement criteria. Event horizons should be exami determine which applications are likely to fail first. Some may only use past dates and w until after the turn of the century. Others using future dates such as expiration dates, are fail much earlier. Additionally, if date expansion is to be considered, all date storage an facilities must also be examined. This includes all files, databases and external interface The impact of expanding the date fields must be analyzed and tied to the application uni accessing these files. Hardware and operating systems must also be considered.

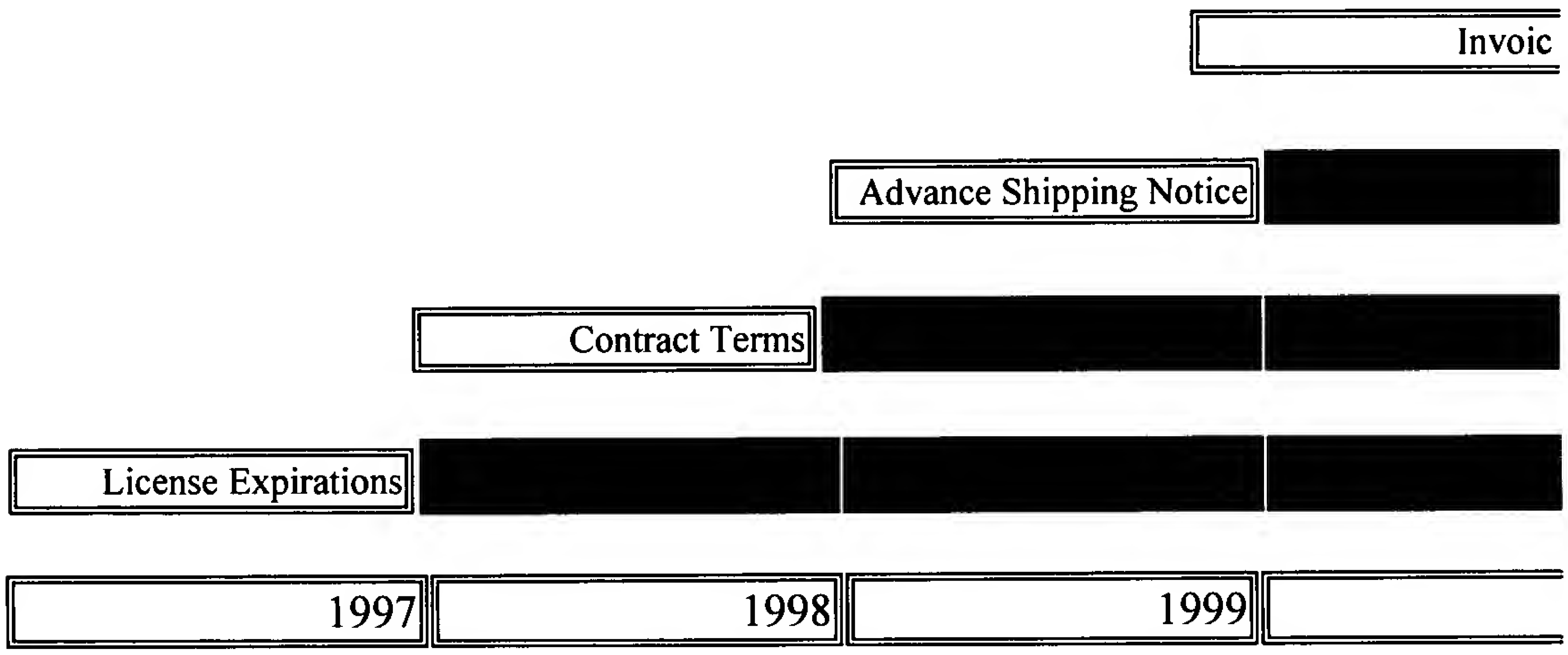
Once an impact analysis is complete, a rough estimate can be made as to the project cos standard software project estimation procedures (e.g., COCOMO) or via a weighted esti process using the information gathered during the analysis phase.

Tools also exist to assist in both the analysis and cost estimation endeavors. (See appen

Return to Top

Prioritize System Applications and Schedule

Once the analysis is completed, prioritization must occur. This is required not only to de starting point but to also decide if triage is necessary. Each application should be consid its contribution to core business operations and revenue generation. Those that rank hig this list are the most critical to the continued success of the business and should be addr first. However, event horizons must also be reviewed and considered. If one critical app not affected until after 2000 and another is affected beginning in 1998, the earlier affect application should be addressed first.



Since it is possible that not all application can be addressed in the time remaining, triag need to be applied to the final priority list. A preliminary modification schedule should determined. Application which fall outside the available time and resource limitations w be reviewed and a decision made as to keep or retire and eliminate. This can be a diffic decision, but may be required if the priority list of applications is long.

Return to Top

Establish Change Management Procedures

Since such a large number of application units will be affected, change management pro and policies must be established and adhered to. If this is not performed, software modi can be a major ordeal and could introduce costly delays into the overall project schedul especially if concurrent development continues during the life of this project. Tools are available to establish software modification procedures and assist in the tracking of soft while in development. Developers must adhere to the rules and guidelines put into place

Return to Top

Establish Application/Group Change Requirements and Boundary Plan

Once applications or software groupings have been prioritized and a tentative schedule established, individual group change requirements can and must be determined. Each gr should be analyzed to determine the specific changes required to bring these software u year 2000 compliancy. The impact of these changes relative to the applications they we developed for need to be reviewed and any anomalies or problem areas addressed. Data need to be mapped and examined throughout the software unit grouping to determine th of any formatting changes.

The result of this analysis and examination can then be incorporated to create a boundar

The boundary plan will list all data feeds entering and exiting the software unit groupin the feeds between software units within the group, external feeds must also be examine determine which contain date information that will be impacted by the compliance chan then defines the boundary for internal and external impacted data. Data passing in and o require bridging processes to be created until applications on both sides of this boundar 2000 compliant.

Return to Top

Define and Establish Overall Test Plan and Baseline Test Data

An overall methodology for testing should be established early in the undertaking of thi This plan should address general testing techniques that are to be applied to all software groupings or applications, as they become year 2000 compliant. A standardized approa essential in minimizing the testing time. Testing environments must also be defined and established. Additional hardware should be planned for and acquired, if necessary.

Additionally, since part of this test plan should include the verification that software mo did not affect current date processing, a baseline of test data should be created and esta each software unit grouping. All software, prior to conversion, should process this test d produce an output data verification source. Both baseline data sets should then be saved reuse and output comparisons, after all conversion modifications have been complete.

Return to Top

Software Modification

Software modification is very straightforward. Software units must be modified to impl solution approach selected. Anomalies and problem areas are also corrected. Source co recompiled (if applicable) and then submitted to testing.

Return to Top

Test, Verify & Validate

Testing consumes the largest segment of effort required to complete a year 2000 compli project, approximately 50% to 60%. Standard unit testing should always be performed i to parallel and system integration testing. A complete replication of all production envir is recommended to perform these tests and should have been defined in the initial test p

Testing, using the input baseline data, should be performed and the results compared to baseline testing output results. Resulting intermediate and final output date data should Mismatches may be encountered and are acceptable when they can be logically explain (E.g., variances due to system date changes, etc.)

Testing will require data samples that include dates both before and after the turn of the Testing will need to be performed with the system clock set both before and after the tu century. Additionally, it may be desirable to execute some software units during the sys clock's actual rollover at the turn of the century.

All testing should be documented and retained in the event re-testing or re-verification i

Return to Top

Implement & Support

Once testing has been completed and the software verified to function properly with res processing of year 2000 dates, implementation can occur. Depending on the solution ap selected, implementation may need to be performed by application or software groupin simply occur on a software unit level (e.g. - on a program-by-program basis).

Software monitoring and support should also be available in the event than any unfores problems should arise after the software is placed into production. A process for handli problems should also be established.

Return to Top

Pilot Project

In some cases, a pilot project is advised. A small, high priority application or software g should be selected for this initial compliance attempt. Standardized analysis, software modification and testing procedures should be applied and adhered to during this initial project. Pitfalls and stumbling blocks should be noted and resolutions addressed prior t of the subsequent compliance projects.

Return to Top

Client Considerations

Client Systems

This document is intended to address the compliance issues of all applications that resid specific hardware platform for one specific client. Like every client, every computer op system has diversity and unique features. However, integration between different syste not be overlooked and should always be considered when selecting a particular complia approach. Features unique to a particular operating environment should be considered a evaluated. However, this uniqueness must not be allowed to place limitations upon exte systems and/or business practices.

NOTE: This example document was produced for a client's applications existing on 25 Packard's 3000 series of computer systems. Other systems also interface with these syst client/server applications, FTP, and DNS and include IBM AS/400's, HP 9000's, and a of PC's (NT & Windows 95) and other UNIX based servers.

Return to Top

Client Assumptions & Understandings

The following client assumptions and understanding have been made in preparing this d

- Mandated software development policies dictate that all future development shall requir field that is added to a database or file shall be maintained in an ISO 8601 format (YY DD). Additionally, all data feeds entering and exiting existing computer systems shall b to use this same format when a choice is available, unless the date feed are for a purcha software package. In this case, the format will be consistent with that which the package has defined as it's standard.
- All computer operating systems and utilities are or will be compliant before the end of
- Event horizons are all less than 25 years with most being less than 10 years. All are unli

change drastically. Future applications with new and/or different event horizons will be already being, or created as, year 2000 compliant.

- Existing software development standards have been adhered to when retrieving system data.
- Developed fourth generation language applications are all executed directly from sourc will be prior to the initiation of the actual software modification phase of this project.

Approach Recommendations

Overview

In recommending a preferred method of solution in becoming year 2000 compliant, sev factors have been considered and evaluated. The results of these efforts then lead to the recommendations.

Return to Top

Primary Considerations

Following is the list of primary considerations that were influential in the approach sele

- Software change management issues are always and have always been a concern of proj managers, developers and system administrators alike. Concurrent development impose continual issues in the management of software development changes. Minimizing the i this process must be considered.
- Ambiguous meanings in the raw data should be avoided. This not only causes problems applications support personnel but also with applications developers attempting to mak of data they must process.
- Application centric conversion is desired rather than data centric conversion. This prom modification, testing and implementation of software units by logical application rather or data set.
- Bridging between applications should be avoided or minimized, if possible. This will k usage from increasing excessively, will prevent having to re-address software modules bridge needs to be removed, and reduces conversion efforts
- Triage should be minimized or avoided. Business management will have major concern and objections to, the elimination of applications that are helpful to the performance an evaluation of business trends.
- Labor and cost expenditures must be considered in selecting a compliance approach. A that are excessively expensive must be assessed and evaluated against complete system replacements.
- Development standards must be considered. These have been created to keep consisten interfaces appearances and functionality, to minimize maintenance efforts through codi consistencies, and to perpetuate efficient software development and software performan Standards should continue to be adhered to and used for all development projects. For t reasons, a standard year 2000 compliance approach should be considered. If multiple a are required to complete compliance efforts, then standards should be defined to dictate approach is used over another.
- Standardized programming techniques must be considered. Routines should be develop used by all software units in an effort to prevent any 'unique' attempts at performing dat comparisons or evaluations.
- Reprocessing of 'unconverted' historical data can be performed with a minimum amount
- Expanded date formats will ultimately be used industry wide and are desired for use wit external business interfaces. These formats should follow guidelines set by national and international standardization organizations. (ISO, IEEE, etc.)
- Event horizons must be considered and examined. Methodologies that will not function if event horizons are excessive (over 100 years) should be eliminated from the selection

- Software and hardware constraints and capabilities should always be considered. Syste that could assist in the compliance effort should be reviewed and their use considered. of database or file access methods could reduce conversion efforts.
- The relationship of software change requirements to approach solutions must also be co Minimizing the number of software units requiring changes will reduce change manage concurrent development issues. The reductions of software unit changes could benefit considerations discussed above including reduced change management issues, reduced for triage, reduced costs, and assist in attaining application-centric conversion grouping

Return to Top

Secondary Considerations

Other consideration should also be reviewed and examined.

- Business practices which keep the company competitive. The year 2000 compliance eff cannot hinder these practices during the conversion development process and after com completed.
- Critical application processes should always be addressed first. Those with event horizo will be impacted earliest should be brought into compliance ahead of all others.
- External and internal software interface considerations must be evaluated. Will these ex providers and users also become year 2000 compliant? When?
- Resource availability and transition concerns should also be considered when addressin of this size. Since the project deadline is fixed and unmovable, resource planning must future shortages and turnover and the impact this could have on the overall solution sch
- Time constraints must also be considered and scheduling change impact alternatives det early as the amount of time is limited.
- The cost-benefits of converting minimally used applications should be evaluated. If cos are too excessive, the application may need to be discarded.

Return to Top

Recommendation

In considering the above criteria, two approach recommendations are being suggested. would inevitably be used regardless of which solution is selected due to already existing constraints. An optional secondary approach is also available which capitalizes on exist development standards and system features inherent to the HP 3000. However, this recommendation is not included in the reprint of this publication. Please call our offices 933-9669 or email us at GRHelm@GRHelm.com for additional information on the '*ELEMENTARY ADDITION*' approach.

Timeline Windowing

The foremost approach recommendation is to implement the "timeline windowing" tech This approach is recommended due to the relatively short event horizons in the existing and to minimize change management efforts related to concurrent development. Additio approach also minimizes the potential for triage, since not all software units that access require changes. Only software units that contain date comparisons or sorts must be phy changed. This approach also capitalizes on the use of standardized function calls to han date windowing calculation, thereby guaranteeing consistent results in century determin

Return to Top

Technical Description - Timeline Windowing

Method Overview

Timeline windowing can be implemented using several techniques. It is currently the m popular method used in year 2000 compliance projects when the century information is readily available and full date expansion is not feasible. Two sets of numeric ranges or are established which correspond to each possible century a date could reside in. There primarily two techniques used to set these ranges or windows, each of which should be based on the particular usage of the date being evaluated.

Timeline windowing is also likely to be used regardless of what primary compliance me decided upon. Certain interface restraints are bound to come up for which the applicatio century-included date field is not appropriate. Situations that may arise include interfaci external data feeds that may simply never contain century information, even though cen information must be stored in the receiving system. User interfaces such as screens may or restricted for various reasons (e.g. - display space limitations, keystroke minimizatio Retrieving and storing dates under these restrictions would also require the use of a win technique, should century information be required in the final retention format.

Date usage's can be categorized into three classifications: *past normal, future normal* an *transitional* relative to the time at which they are processed or acted upon. (See appendi complete definitions and examples.) The particular usage of a date should always be co when setting window boundaries. Event horizons or date ranges must also be considere setting any window range. This will result in the elimination or minimization of any pos misinterpretations.

The basis of timeline windowing entails establishing a range of two-digit values, which (YY) portion of a date is then compared to and a century determination made. This rang the window of years on a one hundred-year timeline. Any year processed which resides window would be considered a future date or belongs in the higher of two centuries. An outside the window is considered a past date or belongs in the lower of two centuries.

Two different techniques can be used to establish window ranges and implement timeli windowing. Each will provide an accurate method of determining century information i applications with event horizons less than 100 years in length.

Return to Top

Pivot Point Windowing

Pivot Point

Years in 2000 .... | ....Years in 1900

|3/43/43/43/43/43/43/43/43/43/43/43/4+3/43/43/43/43/43/43/43/43/43/43/4+3/43/

00------------------------------25-------------------------------50--------------------------------7

---------- ---------99

As indicated, there are two techniques that can be used to establish windowing ranges. method, known as pivot point windowing, uses a fixed two-digit number to mark the wi transition or crossover point. The other transition point is determined by the actual sequ limitation of two-digit numbers, '00' being the bottom of one range and '99' the top of th By using a single pivot point, logic tests are simplified since programmers can capitaliz sequential ordering of the numbers. Years evaluated which are less than the pivot point inherently greater than or equal to '00'), will be determined to exist in the numerically hi century. Years greater than or equal to this number (and inherently less than or equal to be determined to exist in the numerically lower century. The following table shows the this technique using three different pivot points to decipher 1900 dates from 2000 dates

pivot point examples are given to empathize that in most systems, different pivot point be required depending on the usage of the dates being evaluated.

Pivot Point	Evaluate YY	YYLogic	Result CC
20	20	20 is equal to 20	1920
20	99	99 is greater than 20	1999
20	02	02 is less than 20	2002
20	19	19 is less than 20	2019
50	51	51 is greater than 50	1951
50	98	98 is greater than 50	1998
50	01	01 is less than 50	2001
50	49	49 is less than 50	2049
80	97	97 is greater than 80	1997
80	99	99 is greater than 80	1999
80	00	00 is less than 80	2000
80	03	03 is less than 80	2003

The pivot points should be stored, and maintained, as a variable to allow for simple win adjustments as the future becomes the past. In this manner, the window can be rolled for in time, with time, to avoid another compliance conversion effort when dates begin to cl on the pivot point value. Additionally, the century determination ('19' versus '20') shoul be based off of a variable, or off century information supplied by internal system clocks also prevent a rewrite when the next century needs to be addressed ('2100').

Return to Top

Floating Windows

Floating...*Window*----

|.....Future Dates......|..................Past Dates.......................................Past
Dates....................|....Future Dates......|
|3/43/43/43/43/43/43/43/4+3/43/43/43/43/43/43/43/43/43/43/43/43/43/43/4+3/43/4

00.......................10...50..
...99

The second technique of windowing again involves establishing a range of numbers on t date timeline that is used for comparison with dates being processed. However under thi technique, the range limitations are no longer tied to the sequential high and low points two-digit numbering scheme, 00 and 99 respectively. Two different numbers are used to establish the low and high points of a relative time window. Any year falling inside this is considered to be a future date in the current century or the next. Any year falling outsi range or on the boundaries, above or below it, is considered to be in the past, this centur the previous. These ranges are then able to 'float' with time by, using the current system-supplied date in a formula when calculating this window range. This thereby eliminates future need to manually change range boundaries.

This is accomplished using a formula, which includes two variables. The first is the two representation of the current year, which can be obtained from the computer system's ti clock. The second variable is the term, which represents the amount of years (maximum horizon) in the window range. The following tables contain the formula and an example method in use.

Floating Window Formula
Lower boundary limit: current year + 1 Upper boundary limit: (current + term - 1) mod 100

Floating Window Example
Term: 20 Lower boundary limit: 90 + 1 = 91 Current year: 90 Upper boundary limit: (90 + 20 - 1) mod 100 = 9 Resulting range: 91...99 and 00...09 Therefore if the YY portion of an evaluation date is between 91 and 99, inclusi or between 00 and 09, inclusive then it is in the future (1991...2009), otherwis is in the past (1910...1990).

Using the above values produces the following sample of results:

Year	'Future' Range Low	'Future' Range High
1990	[19]91	[20]09
1994	[19]95	[20]13
1998	[19]99	[20]17
2002	[20]03	[20]21
2006	[20]07	[20]25
2010	[20]11	[20]29
2014	[20]15	[20]37

As indicated above, the particular usage of a date should always be considered when set window boundaries in addition to the event horizon. Floating windows are dependent o knowledge of the maximum term of an event horizon, as this is the 'term' value used in t floating window formula.

Again, implementation of this technique should be done with standardized library calls. (s) should include the base date (system date), evaluation date and term value as input parameters. Including the base date gives this technique true 'floating' capabilities as so date may be used in this parameter other than the system date. Furthermore, if horizon t values are subject to change, it would be advisable to store and maintain the term value variable which can be dynamically changed when the term changes.

Return to Top

Conclusion

Either technique of windowing can be used to obtain year 2000 compliance. It is recom

that standardized library routines or calls be always used attain constant window ranges evaluation procedures. This will guarantee constancy throughout all applications and all easy maintenance of date processing routines.

Since this approach does not link the software units to the actual data, a logical applicati grouping or program by program modification and release approach can be used to brin software into year 2000 compliance and production. Additionally, change management minimized since applications and programs can be released independently of one anoth Finally, triage is minimized since not all software will require modifications to continue operate after the year 2000. Furthermore, software that cannot be addressed and having event horizons, will inevitably become fully functional once again after associated dates rolled over into the next century.

Return to Top

Please contact us for additional information on how we can assist you to manage your business into t millennium.

GRHelm@GRHelm.com

Appendix A - Terminology Definitions

Definitions

CCYYMMDD - a date representation format. This format is defined to represent a date with the follow definitions: 'CC' is the millennium and century digits (e.g. - '19' represents a date between January 1, 1 December 31, 1999 inclusive). 'YY' is the decade and year digits (e.g. - 97 represents a date between J and December 31 inclusive of the 98th year of a given millennium and century). 'MM' represents the m number 1 through 12 corresponding to January through December. Finally, 'DD' represents the day of t number 1 through a possible maximum of 31 depending on the month. The date '19970102' would be i to mean January 2nd, 1997.

Event horizon - The maximum time span, in years, that a particular date storage field may contain at an time. This also relates as a term. For example, a 15 year loan term, a 5 year lease term or a 2 year credi activation term. The larger the event horizon, the earlier the applications processing this date field nee year 2000 compliant.

Future normal dates - in CCYYMMDD format, represent dates that are numerically higher than today's when they are normally evaluated or acted upon. In some cases these dates are likely to become transit dates as time moves forward. These dates represent an action or event that would not normally occur f time to come or are perpetually moved forward into the future. Future dates include date of next birthd celebration, new year's eve, some expiration dates, current fiscal period end date, next fiscal period sta and next renewal date.

Past normal dates - in CCYYMMDD format, are dates that are normally either numerically lower than date or equal to today's date and therefore will reside in the past within a matter of hours. These dates represent the occurrence of some action or historical event. Examples of these types of dates include d birth, date of death, last calibration date, last service date, input date, ordered date, billed date, stateme paid date, last price change date and received date to name a few.

Transitional dates - dates that could be either past, present or future dates at the time they are normally evaluated. Therefore, these types of dates can be numerically higher, lower or equal to today's date wh evaluated in CCYYMMDD format. These dates represent an action or event that is about to occur or h occurred. These dates include this year's birthday celebration date, scheduled service date, appointmen ship date, payment due date, effective date, price change date, arrival date and departure date are some examples.

Year 2000 compliance - The ability to differentiate between centuries when century differentiation is r for accurate, complete and comprehensive retrieval date processing. Additionally, to be compliant, sys applications must be able to properly process date information before, during, and after January 1, 200 any interruptions or operational changes that can be associated with the advent of the new century. Fin compliant applications can identify and process leap year(s) correctly, specifically the year 2000 as a l

Return to Top

Appendix B - Tools

Software Tools

HP 3000 Tools

•Time Shift 2000 - Date Data Analysis
•HourGlass 2000 - System Clock Interception
•Time Shift 2000 - Date Data Aging

Please contact us for additional information on tools that are available to assist in completing your yea 2000 efforts.

(If you would like your tools included in this list, please forward your product description and contact information to GRHelm@GRHelm.com

Return to Top

[Home] [Overview] [Year 2000] [Feedback] [Links]

G. R. HELM, INC.

G.R. Helm Information Services
4993 Golden Foothill Parkway
El Dorado Hills, CA 95762
Phone: 916.933.9669 or 888.999.7432
Fax: 916.933.9696
info@grhelm.com

© 1998-99 by G.R. Helm Information Services, Inc. All rights reserved.

Anton Fetting

ADVANCED STRUCTURED COBOL

BATCH, ON-LINE, AND DATA-BASE CONCEPTS

ADVANCED STRUCTURED COBOL

BATCH, ON-LINE, AND DATA-BASE CONCEPTS

TYLER WELBURN

MAYFIELD PUBLISHING COMPANY

MP

MITCHELL PUBLISHING, INC.

To Karen Richardson,

the **K R** of **s**, for her *c*, *w*, and *v*.

Copyright © 1983 by Mayfield Publishing Company
First edition

All rights reserved. No portion of this book may be reproduced in any form or by any means without written permission of the publisher.

COBOL Is an industry language and is not the property of any company or group of organizations. No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein—FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC® I and II, Data Automation Systems, copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell—have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

Library of Congress Catalog Card Number: 82-073737
International Standard Book Number: 0-87484-558-0

Manufactured in the United States of America
Mayfield Publishing Company
285 Hamilton Avenue
Palo Alto, California 94301
(415) 326-1640

Sponsoring editors: Chuck Murphy and Steve Mitchell
Technical editor: Sondra Wallace
Manuscript editor: Claire Comiskey
Managing editor: Pat Herbst
Text designer: Nancy Sears
Cover designer: Barbara Ravizza
Art director and project coordinator: Lawrence Peterson
Technical artist: Pat Rogondino
Production manager: Cathy Willkie
Compositors: Acme Type Company and Frank's Type
Printer and binder: George Banta Company

C LOGIC FOR COMMONLY NEEDED APPLICATION ROUTINES

Certain routines are commonly called for within commercial application systems. This chapter presents a potpourri of background information, concepts, or logic for a number of these routines.

Calendar Routine Logic

When Rome emerged as a world power, the Roman calendar had 12 months: four months with 31 days, seven with 29 days, and February with 28 days, which resulted in a year of only 355 days. To bring the calendar into line with the seasons, the ruling priests added an extra month from time to time. As elected politicians, the priests tended to proclaim a long year when the other elected officials were of their own party and a short year when they were not.

By the time Julius Caesar came to power, the calendar was 80 days behind the sun. Caesar put an end to these calendar capers by imperial decree and made the solar year (365 days and 6 hours) the basis of the calendar in 46 B.C. Although it would have been ideal to form seven 30-day months and five 31-day months, Caesar made February shorter because the Romans thought it to be unlucky. This resulted in the current arrangement of month/day combinations. To handle the extra 6 hours, a leap year was declared for every fourth year. This system, named for Caesar, became known as the **Julian calendar**.

However, the actual length of the solar year is 11½ minutes shorter than the computation that Caesar used. Thus the Julian year is slightly too long; by the sixteenth century the calendar had slipped 10 days from the equinox. Pope Gregory XIII remedied this situation in 1582. After bringing the calendar year even with the solar system once again, he eliminated three out of four centesimal leap years. Thus 1600 was a leap year and 2000 will be a leap year, but 1700, 1800, and 1900 were not. This **Gregorian calendar** is accurate enough for most purposes and is still in use today.

Gregorian dates can be represented in several formats. In the United States, they are typically expressed in **mm/dd/yy** format (*mm* = month number from 01 to 12; *dd* = day number from 01 to 31; and *yy* = year number of the century from 00 to 99). In England and parts of Europe, dates are expressed in **dd/mm/yy** format. In the United States military, dates are commonly recorded as **dd mmm yyyy** (the month is represented as a three-letter month abbreviation, and all four digits of the year are usually recorded).

Within data-processing systems, ANS date representation calls for Gregorian dates to be recorded in either **yymmdd** or **yyyymmdd** format. This format is actually the most logical one because it conforms to normal number-representation conventions in which the most significant digit is positioned as the leftmost digit and the least significant as the rightmost. Because of this positional notation, the ANS format has the advantage of allowing the date to be used directly in calculations, such as finding the number of days between two dates. It also simplifies the process of comparing one date to another to determine which one is earlier or later.

Date computations are simplified even further by Julianizing the date. A **Julianized date** is one in which the year digits are represented on the left and a 3-digit sequential-day number (from 001 to 365, or 366 for a leap year) is recorded on the right. Thus, in accordance with the ANS standard, the date is

stored in **yyddd** or **yyyyddd** format. The obvious advantage to a Julianized date is that date-span computations can be made without need for adjustments when dates are not within the same month.

Most programmers incorrectly call a Julianized date a "Julian" date. A **Julian** date is rarely encountered outside the science of astronomy. Because of the accuracy and compatibility problems inherent in other calendars, astronomers use a **Julian Day Calendar** in which days are numbered sequentially from an arbitrarily selected point in the year 4713 B.C. Thus July 4, 1776, has a Julian Day of 2,369,905.

Date Validation

Figure C.1 shows date-validation logic in an annotated pseudocode form. Checking a date for reasonableness, as shown in step A, is optional but recommended for most applications. Input transactions are typically dated near the processing date. The further a date is from the current date, the greater the likelihood that a date error has been made. Thus it is wise to establish a span of dates as a range check of reasonableness to identify distant entries. Although the choice of such a span may be somewhat arbitrary, it should be based upon the application requirements and processing schedule. Usually, the length of time permitted for past dates will be greater than that specified for future dates. For example, a sales transaction record dated for a prior month may be reasonable whereas one dated a week into the future may be suspect. Recognize that, whenever a check for reasonableness is made, provision must be made for a reentry override to allow valid exception conditions to be accepted back into the system.

Step B of the date-validation logic, in which 2-digit year numbers are converted to 4-digit year values, is not always done in practice but really should be done now that the beginning of a new century is near and adequate storage space is generally available. Countless programs have been written that test whether a 2-digit year is a leap year by dividing by 4 and, if the remainder is zero, consider the year to be a leap year. When the 2-digit year for the leap year 2000 (00) is encountered, the quotient of such a division will be zero and the remainder will be 4; hence the year will incorrectly be considered to be a common year rather than a leap year. Testing two dates for the earlier or later date is another commonly required date-handling routine that can fail when 2-digit year numbers are used and the dates are in different centuries.

Date-validation data fields:

```
DATE (mcyymmddddd)
    YEAR (mcyy)
        MILLENIUM-DIGIT (m)
        CENTURY-DIGIT (c)
        YEAR-DIGITS (yy)
    MONTH (mm)
    GREG-DAY (dd)
    JUL-DAY (ddd)

LEAP-YEAR-SW (yes/no)

VALID-MONTH-SW (yes/no)

VALID-DAY-SW (yes/no)

PRIOR-REASONABLE-DATE (mcyymmdd)

FUTURE-REASONABLE-DATE (mcyymmdd)

MAXIMUM-YEAR-DAYS (ddd)

DATE-EVALUATION-FLAG (valid/invalid/unreasonable)
```

Table of Maximum Days in Each Month

Month	MAXIMUM-DAYS
January	31
February	dd
March	31
April	30
May	31
June	30
July	31
August	31
September	30
October	31
November	30
December	31

Figure C.1. Date-validation logic.

continued

Gregorian date-validation logic:

```
Step A.  Establish reasonableness parameters in relation to
         current date.

           1.  Establish or compute PRIOR-REASONABLE-DATE.

           2.  Establish or compute FUTURE-REASONABLE-DATE.

Step B.  If not recorded, establish century digits of date to be validated.

           1.  If CENTURY-DIGIT not recorded (YEAR less than 0100),
                  set CENTURY-DIGIT to correct value.

           2.  If MILLENIUM-DIGIT not recorded (YEAR less than 1000),
                  set MILLENIUM-DIGIT to correct value.

Step C.  Determine if year to be validated is a leap year.

           1.  If YEAR / 4 produces a remainder equal to 0
                  set LEAP-YEAR-SW to "yes"
               else
                  set LEAP-YEAR-SW to "no".

           2.  If LEAP-YEAR-SW is equal to "yes"
               and YEAR-DIGITS are equal to zero
               and YEAR / 400 produces a remainder not equal to 0
                  reset LEAP-YEAR-SW to "no".

Step D.  Validate month.

           1.  If MONTH is not numeric
               or MONTH is less than "01"
               or MONTH is greater than "12"
                  set VALID-MONTH-SW to "no".

Step E.  Validate day.

           1.  If GREG-DAY is not numeric
               or GREG-DAY is less than "01"
                  set VALID-DAY-SW to "no"
               else
                  set VALID-DAY-SW to "yes".

           2.  If LEAP-YEAR
                  set MAXIMUM-DAYS (2) to "29"
               else
                  set MAXIMUM-DAYS (2) to "28".

           3.  If GREG-DAY is greater than MAXIMUM-DAYS (MONTH)
                  reset VALID-MONTH-SW to "no".

Step F.  Evaluate validity and reasonableness.

           1.  If VALID-MONTH-SW is equal to "no"
               or VALID-DAY-SW is equal to "no"
                  set DATE-EVALUATION-FLAG to "invalid"
               else
                  set DATE-EVALUATION-FLAG to "valid".

           2.  If DATE-EVALUATION-FLAG is equal to "valid"
               and DATE is less than PRIOR-REASONABLE-DATE
                  set DATE-EVALUATION-FLAG to "unreasonable".

           3.  If DATE-EVALUATION-FLAG is equal to "valid"
               and DATE is greater than FUTURE-REASONABLE-DATE
                  set DATE-EVALUATION-FLAG to "unreasonable".
```

Julianized date-validation logic:

```
Step A.  Establish reasonableness parameters in relation to
         current date.  [Same as for Gregorian date validation.]

Step B.  Establish century digits of date to be validated if not recorded.
         [Same as for Gregorian date validation.]
```

Figure C.1. (continued)

```
Step C.   Determine if year to be validated is a leap year.
          [Same as for Gregorian date validation.]

Step D.   Validate month. [Not applicable to Julianized date validation.]

Step E.   Validate day.

           1.   If JUL-DAY is not numeric
                or JUL-DAY is less than "001"
                      set VALID-DAY-SW to "no"
                else
                      set VALID-DAY-SW to "yes".

           2.   If LEAP-YEAR
                      move "366" to MAXIMUM-YEAR-DAYS
                else
                      move "365" to MAXIMUM-YEAR-DAYS.

           3.   If JUL-DAY is greater than MAXIMUM-YEAR-DAYS
                      reset VALID-DAY-SW to "no".

Step F.   Evaluate validity and reasonableness.
          [Same as for Gregorian date validation.]
```

Figure C.1. (continued)

Date-conversion data fields:

```
GREG-DATE (mcyymmdd)
    GREG-YEAR (mcyy)
        GREG-MILLENIUM-DIGIT (m)
        GREG-CENTURY-DIGIT (c)
        GREG-YEAR-DIGITS (yy)
    GREG-MONTH (mm)
    GREG-DAY (dd)

JUL-DATE (mcyyddd)
    JUL-YEAR (mcyy)
        JUL-MILLENIUM-DIGIT (m)
        JUL-CENTURY-DIGIT (c)
        JUL-YEAR-DIGITS (yy)
    JUL-DAY (ddd)

LEAP-YEAR-SW (yes/no)
```

Julianized Day Conversion Table

Month	MONTH-NBR	DAYS-BEFORE
January	01	000
February	02	031
March	03	059
April	04	090
May	05	120
June	06	151
July	07	181
August	08	212
September	09	243
October	10	273
November	11	304
December	12	334

Gregorian date to Julianized date conversion:

```
1.  Move GREG-YEAR to JUL-YEAR.

2.  Move DAYS-BEFORE (GREG-MONTH) to JUL-DAY.

3.  Add GREG-DAY to JUL-DAY.

4.  If LEAP-YEAR
    and GREG-MONTH is greater than "02"
          add 1 to JUL-DAY.
```

Julianized date to Gregorian date conversion:

```
1.  Move JUL-YEAR to GREG-YEAR.

2.  If LEAP-YEAR
    and JUL-DAY is greater than "060"
          subtract 1 from JUL-DAY.

3.  Find first DAYS-BEFORE value equal to or greater than JUL-DAY.

4.  Subtract 1 from corresponding MONTH-NBR giving GREG-MONTH.

5.  Subtract DAYS-BEFORE (GREG-MONTH) from JUL-DAY giving GREG-DAY.
```

Figure C.2. Date-conversion logic.

Date-span computation data fields:

```
JUL-EARLIER-DATE (mcyyddd)
    JUL-EARLIER-DATE-YEAR (mcyy)
    JUL-EARLIER-DATE-DAY (ddd)

JUL-LATER-DATE (mcyyddd)
    JUL-LATER-DATE-YEAR (mcyy)
    JUL-LATER-DATE-DAY (ddd)

GREG-EARLIER-DATE (mcyyddd)

GREG-LATER-DATE (mcyymmdd)

YEAR-SPAN (nnnn)

DAY-SPAN (nnn)

YEARS-LEFT (n)
```

Julianized date-span computation:

```
1.  Subtract JUL-EARLIER-YEAR from JUL-LATER-YEAR giving YEAR-SPAN.

2.  Subtract JUL-EARLIER-DATE-DAY from JUL-LATER-DATE-DAY
          giving DAY-SPAN.

3.  If YEAR-SPAN is not equal to zero
          compute DAY-SPAN = DAY-SPAN + (YEAR-SPAN * 365)
          divide YEAR-SPAN by 4
              add quotient to DAY-SPAN
              move remainder to YEARS-LEFT
          divide JUL-EARLIER-YEAR by 4
              subtract remainder from YEARS-LEFT
          if YEARS-LEFT is positive
              add 1 to DAY-SPAN.
```

Gregorian date-span computation:

```
1.  Convert GREG-EARLIER-DATE to JUL-EARLIER-DATE.

2.  Convert GREG-LATER-DATE to JUL-LATER-DATE.

3.  Use the Julianized date-span computation, above.
```

Figure C.3. Date-span computation logic.

Date Conversion

The pseudocode to handle conversion from a Gregorian to a Julianized date and vice versa is presented in Figure C.2. Except for the adjustment that is required for leap years, such conversion is relatively straightforward.

Date-Span Computation

As specified in the pseudocode of Figure C.3, date spans are more easily computed in Julianized date format. Step 3 of the Julianized date-span computation provides the logic that is necessary to figure the date span when the two dates are in different years. Observe that it contains the logic to adjust each span for the appropriate number of extra leap year days.

Day-of-the-Week Computation

The name of the day of the week upon which a date falls can be computed arithmetically by the logic shown in Figure C.4. The principle behind this method is, for a given date, to begin with the century coefficient. This is a number one less than the day value of the first day of the century. Each common year starts one day later than the previous year; a leap year begins two days later. Thus one day is added for each year until the date that is being computed is reached. This, plus one day for each leap year, adjusts to the start of that year. The month coefficient, which adjusts to the start of the given month, is then added. Finally, the day number is added and the sum is divided by 7 to yield the day of the week.

Day-of-the-week computation data fields:

```
DATE (ccyymmddd)
    CENTURY-DIGITS (cc)
    YEAR-DIGITS (yy)
    MONTH (mm)
    DAY (ddd)

DAY-WORK (nnnn)

DAY-VALUE (n)
```

Century Coefficient Table

Century number		CENTURY-COEFFICIENT
0	(First century)	0
100	(Second century)	1
200		2
300		3
400		4
500		5
600		6
700		7
800		8
900		9
1000		8
1100		7
1200		6
1300		5
1400		4
1500	(until 1582)	3
1600		6
1700		4
1800		2
1900		0
2000		6
2100		4
2200		2
2300		0

Month Coefficient Table

Month	MONTH-COEFFICIENT
January	0
February	3
March	3
April	6
May	1
June	4
July	6
August	2
September	5
October	0
November	3
December	5

Day Code Table

Day Code	DAY-NAME
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

Day-of-the-week computation from a Gregorian date:

```
1.  Move CENTURY-COEFFICIENT (CENTURY-DIGITS + 1) to DAY-WORK.

2.  Add YEAR-DIGITS to DAY-WORK.

3.  Add quotient of (YEAR-DIGITS / 4) to DAY-WORK.

4.  Add MONTH-COEFFICIENT (MONTH) to DAY-WORK.

5.  Add DAY to DAY-WORK.

6.  If leap-year and MONTH is less than 3
        subtract 1 from DAY-WORK.

7.  Divide DAY-WORK by 7
        move remainder to DAY-VALUE.

8.  Day-of-the-week = DAY-NAME (DAY-VALUE + 1).
```

Day-of-the-week computation from a Julianized date:

```
1.  Move CENTURY-COEFFICIENT (CENTURY-DIGITS + 1) to DAY-WORK.

2.  Add YEAR-DIGITS to DAY-WORK.

3.  Add quotient of (YEAR-DIGITS / 4) to DAY-WORK.

4.  Add DAY to DAY-WORK.

5.  Divide DAY-WORK by 7
    Move remainder to DAY-VALUE.

6.  Day-of-the-week = DAY-NAME (DAY-VALUE + 1).
```

Figure C.4. Day-of-the-week computation logic.

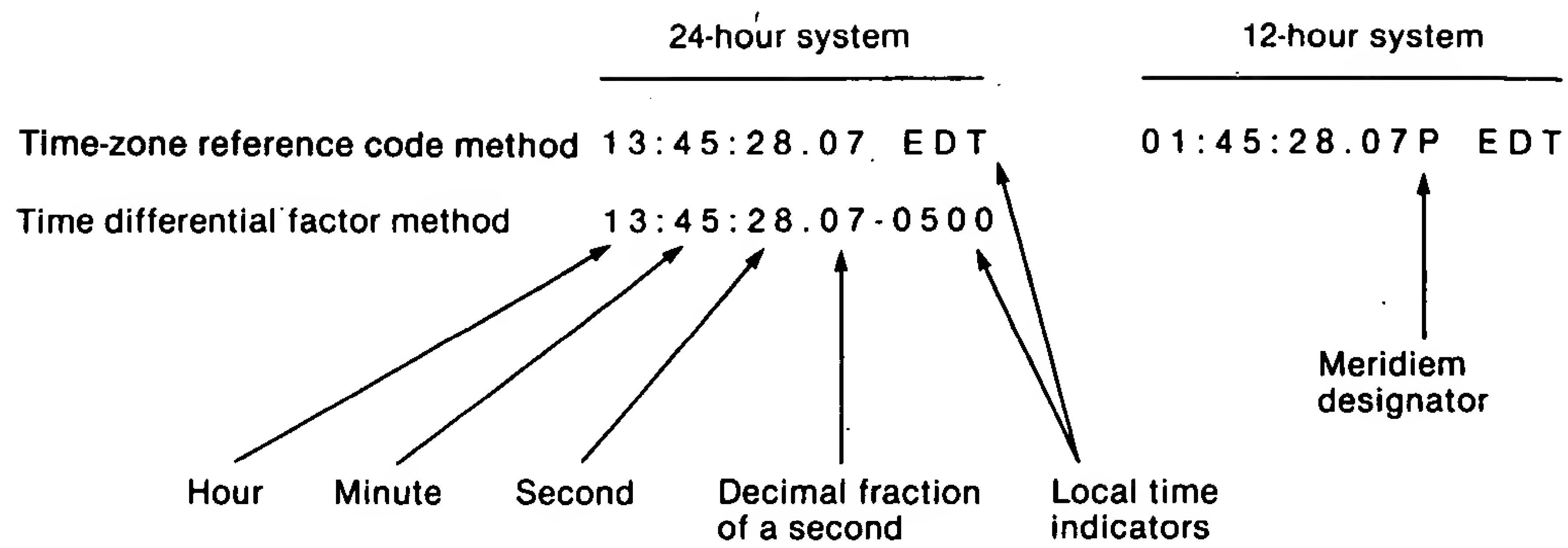


Figure C.5. Time representation.

To simplify the logic, the value of 3 for the majority of the sixteenth century (until 1582) is shown in the century coefficient table. Because the calendar was reformed in October of 1582, a century coefficient of 3 should be used for dates before October 15, 1582; a century coefficient of 0 should actually be used for that and later dates in the sixteenth century.

Time Routine Logic

In everyday life, time is typically expressed in relation to a 12-hour clock. A 24-hour system is used instead in those pursuits—such as the military, astronomy, and aviation—that must frequently convert time from one geographical zone to another. Because of its pervasiveness in the armed forces, the 24-hour system is often called **military time**.

Regardless of 12- or 24-hour representation, time is generally measured in relation to **mean solar time**. Mean solar time is adjusted to **standard time** or **advanced (daylight-saving)** time. Otherwise, a clock on Long Island, for example, would be slightly ahead of one in Newark, New Jersey. The world is divided into 24 time zones, each 15 degrees longitude. There are also a few subzones, such as in Newfoundland, where the time differs from its neighboring zone by only half an hour. For geographical convenience, the time zones sometimes trace political or natural boundaries such as state lines, rivers, and the like.

By tradition, time zones are counted from the Greenwich Observatory in England, which is considered to be the zero meridian. The time zone where the observatory resides is called **Greenwich Mean Time (GMT)**, or **Universal Time (UTC)**. The date changes at the **International Date Line**, which is 12 hours or 180 degrees from Greenwich.

The North American continent spans nine time zones: Newfoundland, Atlantic (easternmost Canada, Puerto Rico, and the Virgin Islands), Eastern, Central, Mountain, Pacific, Yukon, Alaska-Hawaii, and Bering (westernmost Alaska). In accordance with the Uniform Time Act that the U.S. Congress passed in 1966, most of the United States observes daylight-saving time from 2 A.M. on the last Sunday in April until 2 A.M. on the last Sunday in October. During that period, clocks are set ahead one hour. Daylight-saving time is not observed in Arizona, Hawaii, most of Indiana, Puerto Rico, the Virgin Islands, or American Samoa.

As shown in Figure C.5, ANS time representations provide for either a 24- or 12-hour system. The 24-hour system is preferred because it is compatible with the International Standards Organization (ISO) provisions. In addition, it simplifies time conversion and time-span computation tasks.

News

IBM Addresses Year 2000 Crisis

By Joseph McKendrick

November 14, 1995

ARMONK, N.Y. -- Time is running out to re-engineer software to handle the year 2000 date change. A team of experts assembled by IBM has issued a warning that every AS/400, RS/6000, and System/36 system now in use -- along with every other type of computer system -- is at risk of generating faulty data when handling dates for the year 2000 and beyond.

To address the issue, IBM announced it will have current releases of IBM software products year-2000 ready by the end of 1996. The company also announced availability of a free guidebook, along with a set of services, tools and support for the turn-of-the-century transition.

IBM's action stems from the recommendations of a task force consisting of representatives of IBM divisions, IBM's own information technology organization, and a council of 17 major customers, says Charles Lickel, VP of business plans and systems architecture for the IBM Server Group.

For over four decades, developers have written many programs and databases with two-digit year dates -- using "95" rather than "1995." However, when the year date becomes 2000, the computer may interpret 00 to mean 1900. "Obviously, a minor technical change," says Peter de Jager, a Toronto-based consultant. "The trouble is, many companies have more than 50 million lines of code. Where do you start? What do you fix first? Systems are inter-related. Your accounting system feeds information to your inventory system. Inventory is tied to purchasing. Sales is tied to commissions. Commissions is tied to employee benefits. Employee benefits are tied to your pension schemes." De Jager estimates that it would take a single programmer more than seven years to single-handedly fix and test a company's systems.

"This is a cross-platform problem," explains John Phelps of the Gartner Group (Stamford, Conn.). "It has to do with PCs, minis, and mainframes. It's cross-industry -- no one industry is immune, and no one industry is handling the problem better than another. It's cross-vendor."

Phelps cites an example of an insurance company client that gained a competitive advantage by issuing six-year policies, vs. a standard industry practice of issuing three-year policies. "In 1994, he had this cut back to only issuing five-year policies. This year, he only issued four-year policies, and if it's not fixed next year, he'll be back into the three-year policy, with no competitive advantage," Phelps says. Gartner Group estimates that a medium-size company with 8,000 programs will spend between $3.6 and $4.2 million to repair date-challenged software.

The impact is as serious for IBM's midrange-class machines as it is for other types of systems, IBM

executives say. However, there is no problem in AS/400 or RS/6000 hardware. "AS/400 hardware itself handles the problem," says Lickel. "OS/400 Version 3 will handle the year 2000 transition. But customers will have to be on that level of the OS/400 operating system."

A technique called windowing was incorporated into a previous version of OS/400, which IBM executives say will enable applications to work until the year 2040. Windowing defines a specific 100-year interval -- such as 1940 to 2040 -- which can be fixed or sliding. "AS/400 developers saw the year 2000 problem coming, recognized it early, and got a solution out there early," Lickel says. "They chose the windowing technique because it was the less-costly approach to address the issue." However, for the windowing technique to be effective, it requires all programs to use the same assumptions about the date format. These programs may also require annual updates, and the windowing service adds overhead for each date access made, potentially impacting system performance. The next release of OS/400, scheduled by the end of 1996, will provide full-century compatibility, Lickel says.

A renewed push will also be underway to migrate S/36s to a 2000-ready transitional platform. "We will have a transitional offering for S/36 customers as part of this end-of-'96 solution," Lickel says. "There's specific work going on within Rochester to provide a migration path for the S/36 customers, to move them over to an AS/400-like platform." A formal announcement on this system will be made in early 1996, he adds.

Similar challenges are present in RS/6000-based applications. The AIX environment already addresses the issue with routines that support both two-digit and four-digit date fields. Of particular concern are applications that have been recompiled from other platforms to run on RS/6000s, says Lickel.

IBM is offering a comprehensive year 2000 resource guide available at no cost through IBM representatives and through the Internet (http://www.software.ibm.com). The guide includes a list of many widely used IBM products, and spells out the level or levels that will be 2000-ready. Most OS/400-based products will be 2000-ready in their Version 3 releases.

In addition to the customer guidance paper, IBM announced a set of fee-based services to help companies develop year 2000-ready solutions for their systems and applications. Transformation 2000 services, to be delivered by Integrated Systems Solutions Corporation (ISSC), IBM's systems integration arm, will bring together techniques and technologies to facilitate date-field transitions.

While IBM's software will be year 2000-ready, it is uncertain whether third-party applications will be, says Steve Kagan, director of ISSC. An effort is underway to address the inventory and the readiness of IBM's vendor partners. "The ones we've talked to are aware of the problem," Kagan says. "Some have very large portfolios of applications, and have a very significant effort ahead of them to make things 2000-clean."

The IBM executives and consultants agree that there is no silver bullet or single set of tools that will quickly fix the problem. "Planning is the only silver bullet that you're going to find in this whole process," says de Jager.

"This is without a doubt, the largest management challenge most people will face in their lifetime," claims de Jager. "It's unique in that the deadline cannot be moved or missed. It's also unique in that we all share exactly the same deadline. This is not a technical issue."

Copyright © 1997 Boucher Communications, Inc.

| Boucher Communications, Inc. | MIDRANGE Systems | Subscribe NOW |

Welcome to

Bill Schoen's

Y2K Forum On

The Year 2000 and the *Charmar Correction*

QUESTION:
Why was the Y2K allowed to blow up to near world-panic proportions when a simple, known solution existed?

By now many people understand how limited computer capacity in memory and data storage in the 1970s and earlier led to the use of two-digit year dating, the root cause of the Y2K problem. By the early 1980s computer capacity was ample and there was no real justification for continuing to produce non Y2K compliant systems.

At that time a simple correction to system development procedures, making them compliant, would have eliminated about 95 percent of the problem we now face.

An easy and almost cost-free solution, the ***Charmar Correction***, developed by this editor, did exist and was described in the Feb. 13 1984 issue of then the most authoritative journal in the field: ComputerWorld. Based on the concept of prevention, ***Charmar*** introduced the 2-digit year "sliding window" approach and correctly predicted that no simple solution would later be found to fix the problem retroactively--in other words, failure to act promptly would lead to big problems down the road.

This preventive approach was not adopted, and the present crisis is the direct, inevitable and predictable consequence of that failure.

A follow-up story in the Aug. 3, 1998 issue of ComputerWorld updates developments described here and was the motivation for this forum.

Other pages/documents:

Forum
ComputerWorld Feb. 13, 1984 article
ComputerWorld Aug. 3, 1998 article
The Charmar Correction
The Charmar Campaign
Reply by email
Personal data

Charmar Enterprises. Inc.

THE CHARMAR CORRECTION

by

William H. Schoen

Copyright 1984.
All rights reserved.

1.

Charmar Enterprises. Inc.

CONTENTS:

PAGE

1. THE YEAR 2000 PROBLEM: AN EXPLANATION. 3

2. ITS MAGNITUDE. 4

3. WHY A SOFTWARE FIX IS HIGHLY UNLIKELY. 5

4. WHY NOTHING IS BEING DONE. 6

5. THE TIME TO ACT: NOW 7

6. THE FOUR-DIGIT YEAR: PART OF THE SOLUTION. . . . 8

7. THE CHARMAR CORRECTION:

A. THE BASICS. 9
B. RECOMMENDED POLICY. 10
C. THE TWO-DIGIT YEAR DIRECTIVE. 11

8. THE 'YR2000' SUBROUTINE. 12-21

9. THE 'FDYEAR' SUBROUTINE. 22-24

10. LINK INSTRUCTIONS. 25

11. EXHIBITS: A,B,C,D,E. 26-30

2.

Charmar Enterprises. Inc.

THE YEAR 2000 PROBLEM:

THE SERIOUS PROBLEM

IGNORED BY THE ENTIRE DATA PROCESSING COMMUNITY!

As this is being typed, the date is:. . . . . . .12/14/83.

16 years, 17 days later, the date will be:. . . .12/31/99.

The following day's date will be: . . . . .01/01/00.

Take another look at that date -- 01/01/00 -- doesn't it look a bit odd? Exactly what year is it? Is it the year 2000, the year 1900, or some other year?

"By that time", You might say, "I'll just assume it means the year 2000."

Congratulations. You're smarter that any computer that's ever been made.

Unfortunately, almost all of the software written up to now in COBOL, PL/I, and most other languages has neglected to include logic to process 21st Century dates properly. If nothing is done about this, when the year 2000 arrives, you will be unable to enter it into your systems so that your system will recognize that it is greater than the year 1999, and a great deal of your software either won't run or will produce unpredictable results.

This predicament is due to the fact that the standard DP practice has been to maintain two-digit year dates on data files, rather than four-digit year dates.

Year 2000 data, when entered, will have to be represented by the two-digit year "00" which will compare and sort as being less than all 20th Century dates previously entered, as well as causing other difficulties. See EXHIBIT A for details.

3.

Charmar Enterprises. Inc.

THE MAGNITUDE OF THE PROBLEM

If this problem is allowed to progress unchecked until the year 2000, the consequences will be disastrous.

How could any company operate if its accounts receivable, payroll, and

billings systems won't function? If its online modules will not accept the current date? If its production and inventory systems are down?

Without a doubt, any company or institution that fails to take corrective action early on will be in severe trouble, because there is no easy way to fix this problem on a short-term basis.

A company with even a moderately sized library of personalized software would have to spend a fortune, millions of dollars, to correct this problem on a short-term basis, and it could not be done quickly.

Your company probably has at least 50 production programs for every programmer you employ. If 50% of these programs had to be individually modified (figuring $300 per modification -- 10 hours at $30/hr., a very conservative estimate), it would cost your company roughly $7,500 per programmer for program modifications alone, and additional file update programs and JCL modifications would be necessary for external sorts.

Divide the number of programmers your company employs by 120 and that's roughly how many million dollars you'd have to spend to fix this problem now, and it would take you a minimum of six months to do it, assuming half of your programming staff worked full-time on nothing else.

And you are most certainly not alone. This serious problem exists not only in virtually every sizeable American company and civic or governmental institution, its pervasiveness is WORLD-WIDE in scope.

Has an unrecognized business problem of greater magnitude ever existed previously?

Charmar Enterprises. Inc.

WHY A SOFTWARE FIX IS HIGHLY UNLIKELY:

The mind-boggling complexity of modifying programs and JCL to process year 2000 data makes this problem prohibitively difficult to handle by software.

To begin with, there presently are no industry standard naming conventions. How could a piece of software tell which fields within a program's data division are date fields? It certainly couldn't assume that a 9(5) COMP-3 field is a date field. What would be the key?

Similarly, one can't distinguish, by looking at JCL and control cards, which sorts have dates as control fields. This sort control card field specification: SORT FIELDS=(1,3,PD,A) may or may not be a date. There is no way to tell.

In order to revise programs to process year 2000 data, a program's logic must be deciphered, working-storage fields physically altered, arrays reworked, and so on.

A great many companies have their own idiosyncrasies, such as using single digit month fields and embedding date representations within product numbers.

Consider that no two programmers code the same way, that coding standards at different installations vary greatly, and that, if nothing is done, there will be MILLIONS upon MILLIONS of different programs in many different languages needing modifications by the year 2000.

The potential input combinations for a piece of software would be virtually infinite.

For these reasons, it is improbable that a software package could be designed to handle this task. Even if it were, it probably would be extremely expensive, and there would still be the expense of reprocessing most of your software.

Fortunately, the matter is now academic, because an effective preventive plan of action -- complete with the necessary "tools" -- is now available at almost no cost.

Companies that employ the Charmar Correction will have no need for such a software package.

5.

Charmar Enterprises. Inc.

WHY NOTHING IS BEING DONE:

(or Don't Blame Your DP Manager)

All data processing managers share a common condition: they are on the firing line and swamped with work. Most installations have years of work backlogged.

At any one time, a DP manager has a score of people on his back demanding priority attention for their pet projects.

Therefore data processing is a rough and ready, high-pressure process in which only the most pressing problems are attended to. "If it works, don't mess with it" is an axiom of data processing.

The year 2000 problem is, or should be, the proverbial exception to the rule.

At the moment, few if any non-DP managers have even an inkling of the problem.

A competent DP manager will be aware that handling year 2000 data might be difficult. But since nobody who matters is screaming about it, to him the problem is highly deferrable.

Because he hasn't been confronted by an urgent need to analyze this problem, he's probably unaware that:

1. A software fix is unlikely; and

2. The best way to attack this problem is through preventive action now.

Because of this, companies, even the high temples of data processing, continue to perpetuate the problem.

THE TIME TO ACT: NOW.

Current program longevity patterns indicate that if you implement the Charmar Correction during 1984, more than 97% of your production programs would be year-2000-compatible by the arrival of the year 2000 without having to have been modified to make them so.

If, however, you waited until the year 1990 to implement the Charmar Correction, only about 75% of your programs would be year-2000-compatible by the year 2000 without needing modification, in other words, 25% of all your programs would have to have been modified, a major waste of time and money.

Depending on the size of your program library, failure to act promptly can cost you thousands or millions of dollars -- more and more the longer you delay.

If your primary programming language is COBOL, you may test this finding by scanning your production program libraries for the literal "DATE-WRITTEN" to compile percentages of existing programs more than 10 or 16 years old.

The relevant question is this: why should your company continue to write programs that are not year-2000compatible when, after a brief initial realignment, it is just as easy to write programs that are?

Obviously the time for the Charmar Correction is now.

Why wait?

7.

Charmar Enterprises. Inc

THE FOUR-DIGIT YEAR:

PART OF THE SOLUTION.

The "right" way to solve the year 2000 problem would be to require that all new programs contain a four-digit year in every date field, and to devise a way for existing (two-digit year) systems to interface with the new systems.

In a four-digit year system, the year 2000 would compare as being greater than the year 1999, program modifications would be unnecessary, and sorting by date would pose no problem.

Unfortunately, the two-digit year date is too firmly entrenched in existing systems and data files for this to be practical. New programs usually have to refer to existing data files, all of which contain two-digit year dates. Conversion programs would be necessary for sequential files. Duplicate includes or copy members for file formats would be necessary, and so on.

For these reasons, it is impractical to require that all new programs use four-digit year date fields.

However, there are good reasons for requiring that all new data files created by your company have four-digit year date fields, one being that these fields may eventually accumulate data spanning a century in duration, a built-in limitation of the two-digit year date field.

So much for the general background. Now let's move to the solution and how to apply it at your installation.

8.

Charmar Enterprises. Inc.

THE CHARMAR CORRECTION:

THE BASIC PRINCIPLE:

SOLUTION BY ATTRITION.

Require that, effective immediately, all new programs and JCL must include the necessary logic to process 21st Century dates, and the year 2000 problem will virtually solve itself through attrition by the year 2000.

THE TWO BASIC CORRECTIVE ACTIONS:

1. Require that all new programs written using existing two-digit year date data call the 'YR2000' subroutine for comparisons of dates, as well as date editing, conversions, and elapsed time calculations.

2. Establish the policy that all new data files will contain four-digit year date fields. Use the 'FDYEAR' subroutine for these fields.

9.

Charmar Enterprises. Inc.

RECOMMENDED POLICY

FOR THE YEAR 2000 PROBLEM:

Your company's policy for the year 2000 problem should:

1. Implement the Two-Digit Year Directive (which appears on the following page) utilizing the 'YR2000' subroutine. This procedure addresses the bulk of the problem by correcting current -- and harmful-- standard programming practices based on the two-digit year date field.

2. Require that all new data bases and data files created contain four-digit year date fields. Use the 'FDYEAR' date subroutine for these fields.

3. Forbid the purchase of new programming languages or software packages that are not year-2000-compatible.

4. Identify all non-standard date processing idiosyncrasies existing within your company' a DP environment (i.e. systems using product numbers containing date representations, etc.). Determine which of these conditions are (a) not year-2000-compatible and (b) likely to exist by the year 2000. Analyze each for corrective measures and implement them forthwith so that these problems will not continue to be compounded.

THE TWO-DIGIT YEAR DIRECTIVE:

SPECIFIC STEPS FOR REMEDYING

THE YEAR 2000 PROBLEM IN TWO-DIGIT YEAR SYSTEMS.

1. Require that all new programs written contain logic to process year 2000 data. For languages capable of executing a COBOL subroutine (COBOL, PL/I, etc.), this can be accomplished by (a) implementing the 'YR2000' subroutine and (b) distributing the "NEW DATE PROCESSING POLICY" sheet (EXHIBIT B) to all programmers and enforcing compliance to it.

2. A few of your existing date fields may contain dates earlier than the 'YR2000' subroutine default value. Compile a list of these fields and their override values for distribution. (EXHIBIT C).

3. Discontinue writing new programs in other languages until they have been made year-2000-compatible.

4. When sorting records by date, employ a user-exit routine so that revisions will not be needed by the year 2000. (This will only be necessary until the providers of sort software wake up and incorporate this badly needed facility into their packages. See EXHIBIT D.)

5. Consistently mark new year-2000-compatible programs and JCL so they are easily identifiable as such. See EXHIBIT E.

6. Over the years, regularly monitor the volume of "old" programs and JCL remaining. If and when it becomes advisable, you may want to:

A. Establish the policy that whenever an "old" program or JCL stream happens to require modification, it must be made JCL year-2000-compatible.

B. Actively begin modifying the remaining "old" programs and JCL on a systematic basis so that by the year 2000 all of your software will be year-2000-compatible.

7. Finally, prior to the arrival of the year 2000, simulate test runs of your vital systems with year 2000 input to discover and correct any remaining problems.

Year 2000

David E. Whitney

#21

IBM Corporation
Software Design Center
Poughkeepsie, NY
USIB5T7P @ IBMMAIL

Installation code: IBM

SHARE 85
MVS / SCP Project
Session Number: 2822

PRESENTED
FEB 26-MAR 3, 1995
SHARE IN LA, CA

August 1995

PRESENTED
AUG 13-AUG 18, 1995
SHARE IN ORLANDO, FL.

DOCUMENT DISTRIBUTED TO ALL SHARE MEMBERS
VIA CD ROM

Approaches to fixing problems once found ...

1. Change all occurrences of YY to YYYY

   Requires data to change also

   May choose to ignore cosmetic YY

   Requires all applications using the changed data to change simultaneously

2. Change to encode or compress 4-digit year into 2 digit space

   ie. CHAR to packed, decimal to hex, mm/dd/yy to mmddyyyy

   Requires data to change also

   Requires all applications using the changed data to change simultaneously

3. Change code to use a date window with YY

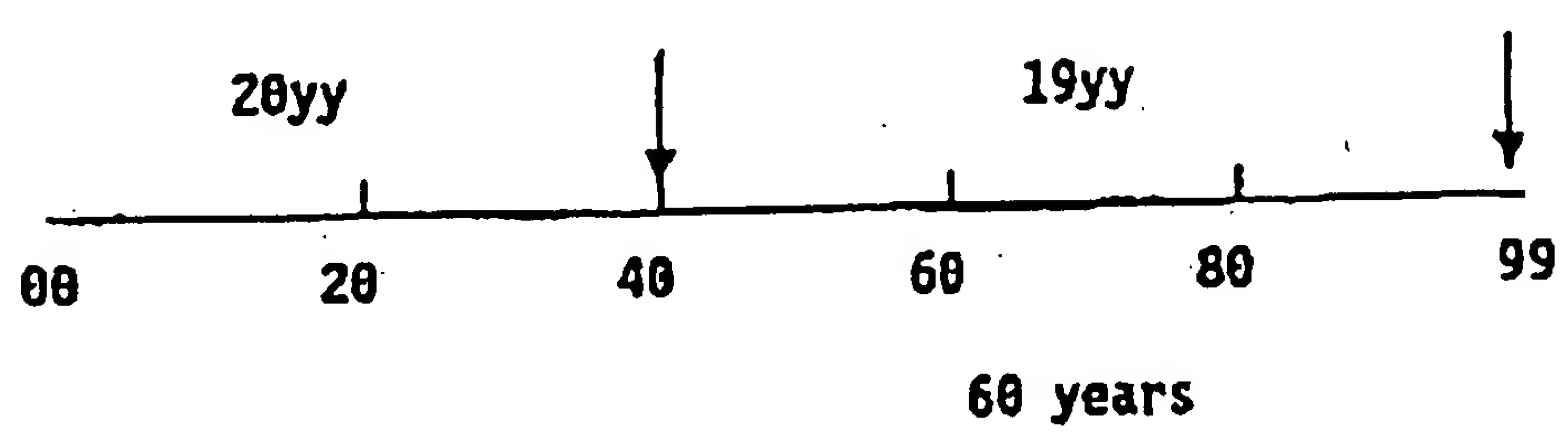
   Requires NO change to data, *but....*

   AD/Cycle LE/370 date routines

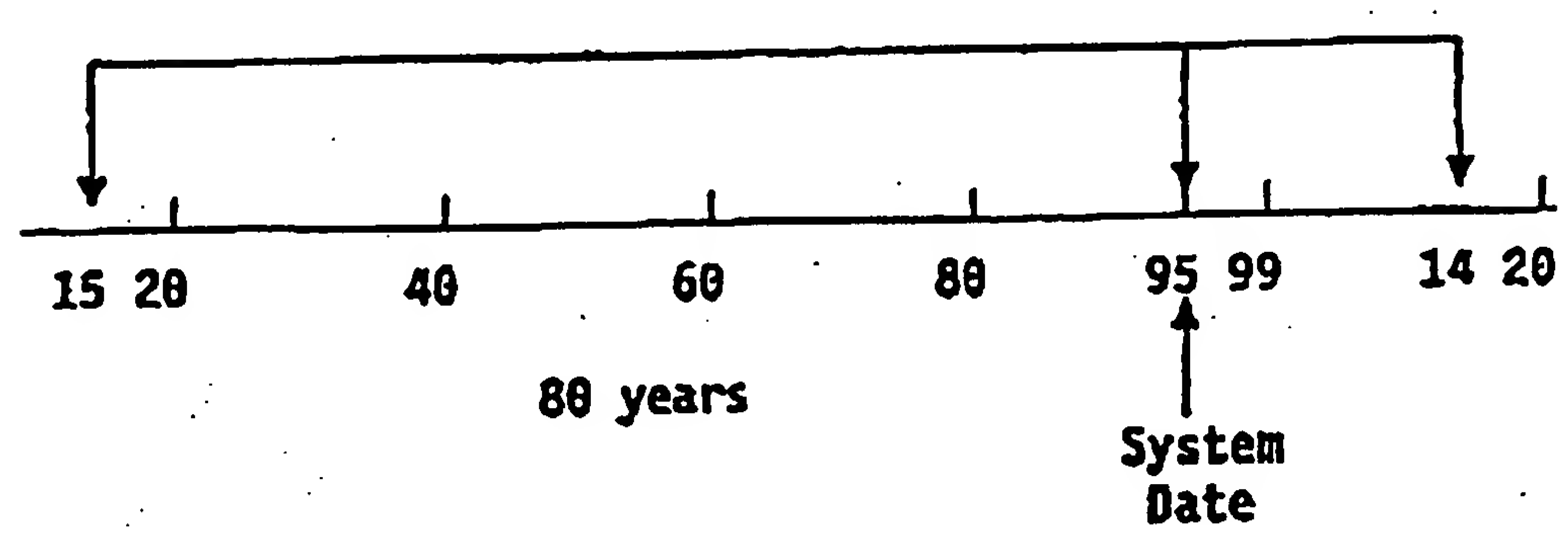
Use of date windows

Change the code to convert YY to YYYY

Fixed window



Sliding window



SHARE 85 CONFERENCE PROCEEDINGS • VOLUME I & II

SHARE Inc.
401 N. Michigan Ave.
Chicago, IL 60611-4267

312/822-0932
312/644-6363 (FAX)
sharehq@share.org
http://www.share.org/

© Copyright SHARE Inc. 1995

OIPE JC127
JAN 0 5 2001
PATENT & TRADEMARK OFFICE

RECEIVED
JAN 0 9 2001
Technology Center 2100

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Reissue Application No.: 09/512,592	
United States Patent No.: 5,806,063	**Group Art Unit:** 2177
Issued: September 8, 1998	**Examiner:** Paul Kulik
Applicant: Dickens-Soeder2000,LLC	**Attorney Docket No.:** 2039-154
Reexamination Proceeding: 90/005,592	
Filed: December 21, 1999	
Reexamination Proceeding: 90/005,628	
Filed: February 2, 2000	
Reexamination Proceeding: 90/005,727	
Filed: May 16, 2000	

RECEIVED
JAN 1 6 2001
OFFICE OF PETITIONS

COPY

HOUSE KEEPING AMENDMENT

Honorable Commissioner of Patents and Trademarks
Washington, D.C. 20231

Dear Sir:

Pursuant to the DECISION, *SUA* SPONTE, TO MERGE REEXAMINATION AND REISSUE PROCEEDINGS, dated November 03, 2000 and mailed November 6, 2000 ("the Decision"),

RECEIVED
JAN 09 2001
Technology Center 2100

the Applicant in the above referenced Reissue Application and Patent Owner in the above referenced Reexamination Proceedings, which were merged by the Decision, hereby submits the House Keeping Amendment called for in the Decision and 37 C.F.R. §1.565(d). This Amendment will serve to place all claims currently in the above referenced Reissue Application in the merged Reexamination Proceeding files. Applicant therefore respectfully requests that the Examiner add the following new claims, the same new claims as were added in the Reissue application, to the above referenced Reexamination Proceeding files. As required by the decision, this identical Amendment is submitted separately in each of the above referenced files, pursuant to the Decision, though these claims are already a part of the above referenced Reissus Application.

RECEIVED
JAN 16 2001
OFFICE OF PETITIONS

In the Claims of the above referenced Reexamination Proceeding files, please add the following new claims:

16. (New) A method of processing symbolic representations of dates stored in a database, comprising the steps of:

providing a database with symbolic representations of dates stored therein according to a format wherein M_1

RECEIVED
JAN 09 2001
Technology Center 2100

M_2 is the numerical month designator, D_1 D_2 is the numerical day designator, and Y_1 Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;

selecting a window with a Y_A Y_B value for a pivot date of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator C_1 C_2 for each symbolic representation of a date in the database, C_1 C_2 having a first value if Y_1 Y_2 is less than Y_A Y_B and having a second value if Y_1 Y_2 is equal to or greater than Y_A Y_B ; and

reformatting the symbolic representation of each symbolic representation of a date in the database, without the addition of any new data field to the database, with the reformatted symbolic representation of each date in the database having the values C_1 C_2, Y_1 Y_2, M_1 M_2, and D_1 D_2, in order to facilitate collectively further processing the reformatted symbolic representations of each of the symbolic representations of each of the dates.

17. (New) The method of claim 16, wherein the window includes at least a portion of the decade beginning in the year 2000.

18. (New) The method of claim 17, wherein the step of determining includes the step of:

determining the first value as 20 and the second value as 19.

19. (New) The method of claim 16, including an additional step, after the step of reformatting, of:

sorting the symbolic representations of dates.

20. (New) The method of claim 16, wherein the step of reformatting includes the step of:

reformatting each symbolic representation of a date into the format $C_1 C_2 Y_1 Y_2 M_1 M_2 D_1 D_2$ separately from the symbolic representations in the database.

21. (New) The method of claim 20, including an additional step, after the step of reformatting, of:

sorting the symbolic representations of dates using a numerical-order sort.

22. (New) The method of claim 16, wherein the step of providing a database includes the step of:

converting pre-existing date information having a different format into the format wherein $M_1 M_2$ is the numerical month designator, $D_1 D_2$ is the numerical day designator and $Y_1 Y_2$ is the numerical year designator.

23. (New) The method of claim 16, wherein the step of selecting includes the step of:

selecting Y_A Y_B such that Y_B is 0 (zero).

24. (New) The method of claim 16, including an additional step, after the step of reformatting, of:

storing the symbolic representation of dates and their associated information back into the database.

25. (New) The method of claim 24, including the additional step, after the step of reformatting, of:

manipulating information in the database having the reformatted date information therein.

26. (New) A method of processing dates in a database, comprising the steps of:

providing a database with dates stored therein according to a format wherein M_1 M_2 is the numerical month designator, D_1 D_2 is the numerical day designator, and Y_1 Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;

selecting a window with a Y_A Y_B value for a pivot date of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator $C_1 C_2$ for each date in the database, $C_1 C_2$ having a first value if $Y_1 Y_2$ is less than $Y_A Y_B$ and having a second value if $Y_1 Y_2$ is equal to or greater than $Y_A Y_B$;

reformatting the symbolic representation of each symbolic representation of a date in the database, without the addition of any new data field to the database, with the reformatted symbolic representation of each date in the database having the values $C_1 C_2$, $Y_1 Y_2$, $M_1 M_2$, and $D_1 D_2$, in order to facilitate collectively further processing the reformatted symbolic representations of each of the symbolic representations of each of the dates; and

sorting the dates in the form $C_1 C_2 Y_1 Y_2 M_1 M_2 D_1 D_2$.

27. (New) The method of claim 26, wherein the step of providing a database includes the step of:

converting pre-existing date information having a different format into the format wherein $M_1 M_2$ is the numerical month designator, $D_1 D_2$ is the numerical day designator and $Y_1 Y_2$ is the numerical year designator.

28. (New) The method of claim 26, wherein the step of selecting includes the step of:

selecting $Y_A Y_B$ such that Y_B is 0 (zero).

29. (New) The method of claim 26, including an additional step, after the step of sorting, of:

storing the sorted dates and their associated information back into the database.

30. (New) The method of claim 29, including the additional step, after the step of sorting, of:

manipulating information in the database having the reformatted dates therein.

31. (New) A method of processing symbolic representations of dates stored in a database, comprising the steps of:

providing a database with symbolic representations of dates stored therein according to a format wherein Y_1 Y_2 is the numerical year designator;

selecting a window with a Y_A Y_B value for the first decade of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator C_1 C_2 for each symbolic representation of a date in the database, C_1 C_2 having a first value if Y_1 Y_2 is less than Y_A Y_B and having a second value if Y_1 Y_2 is equal to or greater than Y_A Y_B ; and

reformatting the symbolic representation of each symbolic representation of a date in the database, without the addition of any new data field to the database, with the reformatted symbolic representation of each date in the database having the values C_1 C_2, Y_1 Y_2, in order to facilitate collectively further processing the reformatted symbolic representations of each of the symbolic representations of each of the dates.

32. (New) A method of processing dates in a database, comprising the steps of:

providing a database with symbolic representations of dates stored therein according to a format wherein Y_1 Y_2 is the numerical year designator;

selecting a window with a Y_A Y_B value for a pivot year of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator C_1 C_2 for each symbolic representation of a date in the database, C_1 C_2 having a first value if Y_1 Y_2 is less than Y_A Y_B and having a second value if Y_1 Y_2 is equal to or greater than Y_A Y_B ;

reformatting the symbolic representation of each of the dates in the database, without the addition of

any new data field to the database, with the reformatted symbolic representation of each date in the database having the values C_1 C_2, Y_1 Y_2, in order to facilitate collectively further processing the reformatted symbolic representations of each of the dates; and

sorting the dates in the form C_1 C_2 Y_1 Y_2.

33. (New) A method of processing symbolic representations of dates stored in a database, comprising the steps of:

providing a database with symbolic representations of dates stored therein according to a format wherein Y_1 Y_2 is the numerical year designator;

selecting a window with a Y_A Y_B value for the first decade of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator C_1 C_2 for each symbolic representation of a date in the database, C_1 C_2 having a first value if Y_1 Y_2 is less than Y_A Y_B and having a second value if Y_1 Y_2 is equal to or greater than Y_A Y_B ; and

reformatting the symbolic representation of each symbolic representation of a date in the database, without changing any of the symbolic representations

of a date in the database during the reformatting step, with the reformatted symbolic representation of each date in the database having the values C_1 C_2, Y_1 Y_2, in order to facilitate collectively further processing the reformatted symbolic representations of each of the dates.

34. (New) A method for representing and utilizing dates stored in at least one date field of a database utilizing symbolic representations of the dates stored in the at least one date field of the database, which are in a format that creates ambiguity between dates in each of a pair of adjacent centuries, comprising the steps of:

converting each of the symbolic representations of dates stored in the at least one date field of the database to a symbolic representation of each of the respective dates that does not create the ambiguity, by windowing the symbolic representations of each of the respective dates as stored in the at least one date field of the database against a pivot year represented by one of the symbolic representations of the dates as stored in the at least one date field of the database, without the addition of any new data field to the database for purposes of such windowing and converting; and,

running a program collectively on each of the converted symbolic representations of each of the respective dates to sort or otherwise manipulate the dates represented by the converted symbolic representations, separately from the date data symbolic representations contained in the at least one date field of the database.

35. (New) A method of claim 34 further comprising the step of:

opening the database prior to the step of converting.

36. (New) The method of claim 34 further comprising the step of:

collectively sorting the converted symbolic representations prior to the step of running the program on the converted symbolic representations.

37. (New) The method of claim 35 further comprising the step of:

collectively sorting the converted symbolic representations prior to the step of running the program on the converted symbolic representations.

38. (New) The method of claim 34 further comprising the step of:

collectively manipulating the converted symbolic representations prior to the step of running the program on the converted symbolic representations.

39. (New) The method of claim 35 further comprising the step of:

collectively manipulating the converted symbolic representations prior to the step of running the program on the converted symbolic representations.

40. (New) The method of claim 34 further comprising the step of:

collectively sorting the converted symbolic representations according to a different data field contained in the database from the at least one date field, prior to the step of running the program on the converted symbolic representations.

41. (New) The method of claim 35 further comprising the step of:

collectively sorting the converted symbolic representations according to a different data field contained in the database from the at least one date

field, prior to the step of running the program on the converted symbolic representations.

42. (New) The method of claim 34 further comprising the step of:

collectively manipulating the converted symbolic representations according to a different data field contained in the database from the at least one date field, prior to the step of running the program on the converted symbolic representations.

43. (New) The method of claim 35 further comprising the step of:

collectively manipulating the converted symbolic representations according to a different data entry field contained in the database from the at least one date field, prior to the step of running the program on the converted symbolic representations.

44. (New) The method of claim 34 wherein the program performs an operation which manipulates the data in a data field associated with the at least one date field of the database according to the converted symbolic representation of the date.

--

45. (New) The method of claim 35 wherein the program performs an operation which manipulates the data in a data field associated with the at least one date field of the database according to the converted symbolic representation of the date.

46. (New) The method of claim 34 wherein the step of converting includes converting at least a substantial portion of each of the plurality of symbolic representations of dates in the at least one date field and repeating this step until each of the date data entries in the at least one date field is converted into the format that does not have the ambiguity.

47. (New) The method of claim 35 wherein the step of converting includes converting at least a substantial portion of each of the plurality of symbolic representations of dates in the at least one date field and repeating this step until each of the date data entries in the at least one date field is converted into the format that does not have the ambiguity.

48. (New) The method of claim 46 further comprising the steps of:

collectively sorting the converted symbolic representations prior to the step of running the program on the converted symbolic representations.

49. (New) The method of claim 47 further comprising the steps of:

collectively sorting the converted symbolic representations prior to the step of running the program on the converted symbolic representations.

50. (New) The method of claim 46 further comprising the step of:

collectively manipulating the converted symbolic representations.

51. (New) The method of claim 49 further comprising the step of:

collectively manipulating the converted symbolic representations.

52. (New) The method of claim 46 further comprising the step of:

collectively sorting the converted symbolic representations according to a different data field in

the database than the at least one date field, prior to the step of running the program.

53. (New) The method of claim 47 further comprising the step of:

collectively sorting the converted symbolic representations according to a different data field in the database than the at least one date field, prior to the step of running the program.

54. (New) The method of claim 52 further comprising the step of:

collectively manipulating the converted symbolic.

55. (New) The method of claim 53 further comprising the step of:

collectively manipulating the converted symbolic representations.

56. (New) The method of claim 52 wherein the program performs an operation which manipulates the data in a data field associated with the at least one date field of the database according to the converted symbolic representation of the date.

57. (New) The method of claim 53 wherein the program performs an operation which manipulates the data in a data field associated with the at least one date field of the database according to the converted symbolic representation of the date.

58. (New) The method of claim 54 wherein the program performs an operation which manipulates the data in a data field associated with the at least one date field of the database according to the converted symbolic representation of the date.

59. (New) The method of claim 55 wherein the program performs an operation which manipulates the data in a data field associated with the at least one date field of the database according to the converted symbolic representation of the date.

60. (New) A method for representing and utilizing dates stored in at least one date field of a database utilizing symbolic representations of the dates stored in the at least one date field of the database, which are in a format that creates ambiguity between dates in each of a pair of adjacent centuries, comprising the steps of:

converting each of the symbolic representations of dates stored in the at least one date field of the database to a symbolic representation of each of the respective dates that does not create the ambiguity, by windowing the symbolic representations of each of the respective dates as stored in the at least one date field of the database against a pivot year represented by one of the symbolic representations of the dates as stored in the at least one date field of the database, without modifying any of the symbolic representations of dates in the at least one date field of the database for purposes of such windowing and converting;

running a program on each of the converted symbolic representations of each of the respective dates to sort or otherwise manipulate data in the database according to the dates represented by the converted symbolic representations, separately from the date data symbolic representations of dates contained in the at least one date field of the database.

61. (New) A method for representing and utilizing dates stored in at least one date field of a database utilizing symbolic representations of the dates stored in the at least one date field of the database, which are in a format that creates ambiguity between dates in

each of a pair of adjacent centuries, comprising the steps of:

converting each of the symbolic representations of dates stored in the at least one date field of the database to a symbolic representation of each of the respective dates that does not create the ambiguity, by windowing the symbolic representations of each of the respective dates as stored in the at least one date field of the database against a pivot year represented by one of the symbolic representations of the dates as stored in the at least one date field of the database, without modifying any of the symbolic representations of dates in the at least date field of the database for purposes of such windowing and converting;

running a program collectively on each of the converted symbolic representations of each of the respective dates to sort or otherwise manipulate the dates represented by the converted symbolic representations, separately from the symbolic representations of dates contained in the at least one date field of the database.

62. (New) A method for representing and utilizing dates stored in at least one date field of a database utilizing symbolic representations of the dates stored

in the at least one date field of the database, which are in a format that creates ambiguity between dates in each of a pair of adjacent centuries, comprising the steps of:

converting each of the symbolic representations of dates stored in the at least one date field of the database to a symbolic representation of each of the respective dates that does not create the ambiguity, by windowing the symbolic representations of each of the respective dates as stored in the at least one date field of the database against a pivot year represented by one of the symbolic representations of the dates as stored in the at least one date field of the database, without the addition of any new data field to the database for purposes of such windowing and converting;

storing the converted symbolic representations separate from the at least one date field of the database; and

running a program on the stored converted symbolic representations to sort or otherwise manipulate data in the database according to the dates represented by the converted symbolic representations, separately from the symbolic representations of dates contained in the at least one date field of the database.

63. (New) A method for representing and utilizing dates stored in at least one date field of a database utilizing symbolic representations of the dates stored in the at least one date field of the database, which are in a format that creates ambiguity between dates in each of a pair of adjacent centuries, comprising the steps of:

converting each of the symbolic representations of dates stored in the at least one date field of the database to a symbolic representation of each of the respective dates that does not create the ambiguity, by windowing the symbolic representations of each of the respective dates as stored in the at least one date field of the database against a pivot year represented by one of the symbolic representations of the dates as stored in the at least one date field of the database, without the addition of any new data field to the database for purposes of such windowing and converting;

storing the converted symbolic representations separate from the at least one date field of the database; and

running a program collectively on the stored converted symbolic representations to sort or otherwise manipulate the dates represented by the

converted symbolic representations, separately from the symbolic representations of dates contained in the at least one date field of the database.

64. (New) A method for representing and utilizing dates stored in at least one date field of a database utilizing symbolic representations of the dates stored in the at least one date field of the database, which are in a format that creates ambiguity between dates in each of a pair of adjacent centuries, comprising the steps of:

converting each of the symbolic representations of dates stored in the at least one date field of the database to a symbolic representation of each of the respective dates that does not create the ambiguity, by windowing the symbolic representations of each of the respective dates as stored in the at least one date field of the database against a pivot year represented by one of the symbolic representations of the dates as stored in the at least one date field of the database, without modifying any of the symbolic representations of dates in the at least one date field of the database for purposes of such windowing and converting;

storing the converted symbolic representations separate from the at least one date field in the database; and

running a program on the stored converted symbolic representations to sort or otherwise manipulate data in the database according to the dates represented by the converted symbolic representations, separately from the symbolic representations of dates contained in the at least one date field of the database.

65. (New) A method for representing and utilizing dates stored in at least one date field of a database utilizing symbolic representations of the dates stored in the at least one date field of the database, which are in a format that creates ambiguity between dates in each of a pair of adjacent centuries, comprising the steps of:

converting each of the symbolic representations of dates stored in the at least one date field of the database to a symbolic representation of each of the respective dates that does not create the ambiguity, by windowing the symbolic representations of each of the respective dates as stored in the at least one date field of the database against a pivot year represented by one of the symbolic representations of the dates as stored in the at least one date field of

the database, without modifying any of the symbolic representations of dates in the at least one date field of the database for purposes of such windowing and converting;

storing the converted symbolic representations separate from the at least one date field in the database; and

running a program collectively on the stored converted symbolic representations to sort or otherwise manipulate the dates represented by the converted symbolic representations, separately from the symbolic representations of dates contained in the at least one date field of the database.

66. (New) A method of processing dates in a database, comprising the steps of:

providing a database with dates stored in at least one date field therein according to a format wherein M_1 M_2 is the numerical month designator, D_1 D_2 is the numerical day designator, and Y_1 Y_2 is the numerical year designator;

selecting a window with a Y_A Y_B value for a pivot date of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator $C_1 C_2$ for each date in the database, $C_1 C_2$ having a first value if $Y_1 Y_2$ is less than $Y_A Y_B$ and having a second value if $Y_1 Y_2$ is equal to or greater than $Y_A Y_B$;

reformatting the symbolic representation of each symbolic representation of a date in a portion of the at least one date field in the database, without the addition of any new data field to the database, with the reformatted symbolic representation of each date in the database having the values $C_1 C_2$, $Y_1 Y_2$, $M_1 M_2$, and $D_1 D_2$; and

repeating the step of reformatting until each symbolic representation of a date in the at least one date field has been reformatted in order to facilitate collectively further processing the reformatted symbolic representations of each of the symbolic representations of each of the dates.

67. (New) A method of processing dates in a database, comprising the steps of:

providing a database with dates stored in at least one date field therein according to a format wherein $Y_1 Y_2$ is the numerical year designator;

selecting a window with a Y_A Y_B value for a pivot date of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator C_1 C_2 for each date in the database, C_1 C_2 having a first value if Y_1 Y_2 is less than Y_A Y_B and having a second value if Y_1 Y_2 is equal to or greater than Y_A Y_B;

reformatting the symbolic representation of each symbolic representation of a date in a portion of the at least one date field in the database, without the addition of any new data field to the database, with the reformatted symbolic representation of each date in the database having the values C_1 C_2, Y_1 Y_2; and

repeating the step of reformatting until each symbolic representation of a date in the at least one date field has been reformatted in order to facilitate collectively further processing the reformatted symbolic representations of each of the symbolic representations of each of the dates.

68. (New) A method of processing symbolic representations of dates stored in a database, comprising the steps of:

providing a database with symbolic representations of dates stored in at least one date field therein

according to a format wherein Y_1 Y_2 is the numerical year designator;

selecting a window with a Y_A Y_B value for the first decade of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the at least one date field of the database;

determining a century designator C_1 C_2 for each symbolic representation of a date in the database, C_1 C_2 having a first value if Y_1 Y_2 is less than Y_A Y_B and having a second value if Y_1 Y_2 is equal to or greater than Y_A Y_B ; and

reformatting the symbolic representation of each symbolic representation of a date in at least one date field in the database, without the addition of any new data field to the database, with the reformatted symbolic representation of each date in the database having the values C_1 C_2, Y_1 Y_2, in order to facilitate further processing of the reformatted symbolic representations of each of the symbolic representations of each of the dates, by running a program on the reformatted symbolic representations of each of the dates.

69. (New) A method of processing dates in a database, comprising the steps of:

providing a database with dates stored in at least one date field therein according to a format wherein $Y_1 Y_2$ is the numerical year designator;

selecting a window with a $Y_A Y_B$ value for a pivot year of the window, $Y_A Y_B$ being no later than the earliest $Y_1 Y_2$ year designator in the database;

determining a century designator $C_1 C_2$ for each date in the at least one date field of the database, $C_1 C_2$ having a first value if $Y_1 Y_2$ is less than $Y_A Y_B$ and having a second value if $Y_1 Y_2$ is equal to or greater than $Y_A Y_B$;

reformatting the symbolic representation of each symbolic representation of a date in the at least one date field in the database, without the addition of any new data field to the database, with the reformatted symbolic representation of each date in the database having the values $C_1 C_2$, $Y_1 Y_2$;

sorting the reformatted symbolic representations of the dates in the form $C_1 C_2 Y_1 Y_2$; and

running a program on the reformatted symbolic representations of each of the dates.

70. (New) A method for representing and utilizing dates stored in at least one date field of a database

utilizing symbolic representations of the dates stored in at least one date field of the database, which are in a format that creates ambiguity between dates in each of a pair of adjacent centuries, comprising the steps of

converting each of the symbolic representations of dates stored in the at least one date field of the database to a symbolic representation of each of the respective dates that does not create the ambiguity, by windowing the symbolic representations of each of the respective dates as stored in the at least one date field of the database against a pivot year, with the pivot year being less than or equal to the earliest date represented by the symbolic representation of dates stored in the at least one date field, without the addition of any new data field to the database, and without modifying any of the symbolic representations of dates in the at least one date field, for purposes of such windowing and converting; and,

running a program on the converted symbolic representations of each of the dates to sort or otherwise manipulate the dates represented by the converted symbolic representations, separately from

the date data symbolic representations contained in the at least one date field of the database.

71. (New) A method for representing and utilizing dates stored in at least one date field of the database utilizing symbolic representations of the dates stored in the at least one date field of the database, which are in a format that creates ambiguity between dates in each of a pair of adjacent centuries, comprising the steps of

converting each of the symbolic representations of dates stored in the at least one date field of the database to a symbolic representation of each of the respective dates that does not create the ambiguity, by windowing the symbolic representations of each of the respective dates as stored in the at least one date field of the database against a pivot year, with the pivot year being less than or equal to the earliest date represented by a symbolic representation of dates stored in the at least one date field, and without the addition of any new data field to the database for purposes of such windowing and converting;

storing each of the converted symbolic representations of each of the dates separate from the database; and,

running a program on the stored converted symbolic representations of each of the converted symbolic representations of the dates to sort or otherwise manipulate the dates represented by the converted symbolic representations, separately from the date data symbolic representations contained in the at least one date field of the database.

72. (New) A method of processing symbolic representations of dates stored in a database, comprising the steps of

selecting a database with symbolic representations of dates stored therein according to a format wherein M_1 M_2 is the numerical month designator, D_1 D_2 is the numerical day designator, and Y_1 Y_2 is the numerical year designator;

selecting a 10-decade window with a Y_A Y_B value for the first decade of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator C_1 C_2 for each symbolic representation of a date in the database, C_1 C_2 having a first value if Y_1 Y_2 is less than Y_A Y_B and having a second value if Y_1 Y_2 is equal to or greater than Y_A Y_B ; and,

reformatting the symbolic representation of each symbolic representation of a date in the database with the values C_1 C_2, Y_1 Y_2, M_1 M_2 , and D_1 D_2 prior to collectively further processing information contained within the database associated with the respective dates.

73. (New) A method of processing symbolic representations of dates stored in a database, comprising the steps of

providing a database with symbolic representations of dates stored therein according to a format wherein Y_1 Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;

selecting a 10-decade window with a Y_A Y_B value for the first decade of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator C_1 C_2 for each symbolic representation of a date in the database, C_1 C_2 having a first value if Y_1 Y_2 is less than Y_A Y_B and having a second value if Y_1 Y_2 is equal to or greater than Y_A Y_B ; and,

reformatting the symbolic representation of the date with the values C_1 C_2, Y_1 Y_2, to facilitate further processing of the dates.

74. (New) A method of processing dates in a database, comprising the steps of

providing a database with symbolic representations of dates stored therein according to a format wherein Y_1 Y_2 is the numerical year designator, all of symbolic representations of dates falling within a 10-decade period of time;

selecting a 10-decade window with a Y_A Y_B value for the first decade of the window, Y_A Y_B being no later than the earliest Y_1 Y_2 year designator in the database;

determining a century designator C_1 C_2 for each date in the database, C_1 C_2 having a first value if Y_1 Y_2 is less than Y_A Y_B and having a second value if Y_1 Y_2 is equal to or greater than Y_A Y_B ;

reformatting each date in the form C_1 C_2 Y_1 Y_2 to facilitate further processing of the dates; and,

sorting the dates in the form C_1 C_2 Y_1 Y_2.

75. (New) A method of processing symbolic representations of dates stored in a database, comprising the steps of

providing a database with symbolic representations of dates stored therein according to a format wherein $M_1 M_2$ is the numerical month designator, $D_1 D_2$ is the numerical day designator, and $Y_1 Y_2$ is the numerical year designator;

selecting a window with a $Y_A Y_B$ value for a pivot date of the window, $Y_A Y_B$ being no later than the earliest $Y_1 Y_2$ year designator in the database;

determining a century designator $C_1 C_2$ for each symbolic representation of a date in the database, $C_1 C_2$ having a first value if $Y_1 Y_2$ is less than $Y_A Y_B$ and having a second value if $Y_1 Y_2$ is equal to or greater than $Y_A Y_B$; and

reformatting the symbolic representation of each symbolic representation of a date in the database, without the addition of any new data field to the database, with the reformatted symbolic representation of each date in the database having the values $C_1 C_2$, $Y_1 Y_2$, $M_1 M_2$, and $D_1 D_2$, in order to facilitate further processing of the reformatted

symbolic representations of each of the symbolic representations of each of the dates.

76. (New) A method of processing dates in a database, comprising the steps of

providing a database with dates stored therein according to a format wherein $M_1 M_2$ is the numerical month designator, $D_1 D_2$ is the numerical day designator, and $Y_1 Y_2$ is the numerical year designator;

selecting a window with a $Y_A Y_B$ value for a pivot date of the window, $Y_A Y_B$ being no later than the earliest $Y_1 Y_2$ year designator in the database;

determining a century designator $C_1 C_2$ for each date in the database, $C_1 C_2$ having a first value if $Y_1 Y_2$ is less than $Y_A Y_B$ and having a second value if $Y_1 Y_2$ is equal to or greater than $Y_A Y_B$;

reformatting the symbolic representation of each symbolic representation of a date in the database, without the addition of any new data field to the database, with the reformatted symbolic representation of each date in the database having the values $C_1 C_2$, $Y_1 Y_2$, $M_1 M_2$, and $D_1 D_2$, in order to facilitate further processing of the reformatted

symbolic representations of each of the symbolic representations of each of the dates; and

sorting the dates in the form $C_1 C_2 Y_1 Y_2 M_1 M_2 D_1 D_2$.

Remarks

The above amendment, pursuant to the requirements of the Decision and 37 C.F.R.§1.565(d), places the claims added to the Reissue Application in the files for the above referenced Reexamination Proceedings.

Respectfully submitted,

William C. Cray
Reg.No. 27,627

ASSISTANT SECRETARY
AND COMMISSIONER

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

2000 JAN -4 PM 3:11
U.S. PATENT
AND
TRADEMARK OFFICE

In re Patent of | Date : December 28, 1999

Bruce Dickens

U.S. Patent No. : 5,806,063

Reexamination No. : (unknown)

Issue Date : September 8, 1998

For : DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

Q. Todd Dickinson
Commissioner of Patents and Trademarks
Washington, D.C. 20231

Sir:

COMMUNICATION

Enclosed is prior art to U.S. Patent No. 5,806,063, entitled DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY, granted to Bruce Dickens of Irvine, California, on September 8, 1998. The enclosed prior art is a photocopy of selected portions of a book titled *The Year 2000 Computing Crisis,* by Jerome T. Murray and Marilyn J. Murray, published in March, 1996 by McGraw-Hill. A printout of the official McGraw-Hill web page for this book also is enclosed.

The enclosed prior art is a printed publication, and is pertinent and applicable to the patent. The enclosed prior art is believed to have a bearing on the patentability of at least one of the claims of the '063 patent, under 35 U.S.C. §§ 102(a) and 103(a).

It respectfully is requested that the enclosed prior art be entered in the patent file, to the extent that the prior art has not yet been entered. It further is requested that entry into the

patent file of this citation not be delayed until the Reexamination proceedings have been terminated.

A copy of this Communication and the enclosed prior art is being sent to the attorney identified below, who is believed to represent Mr. Dickens, and thereby be associated with the patent owner in the present Reexamination proceeding, and be subject to a duty to disclose this information under 37 C.F.R. § 1.55(a).

William C. Cray, Partner
Levin & Hawes
384 Forest Avenue, Suite 13
Laguna Beach, California 92551

Computing McGraw-Hill

THE YEAR 2000 COMPUTING CRISIS

A Millennium Date Conversion Plan

Jerome T. Murray

Marilyn J. Murray

The Year 2000 Computing Crisis

A Millennium Date Conversion Plan

Jerome T. Murray
Marilyn J. Murray

McGraw-Hill
New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

McGraw-Hill
A Division of The McGraw-Hill Companies

Library of Congress Cataloging-in-Publication Data

Murray, Jerome T.
The year 2000 computing crisis : a millennium date conversion plan / by Jerome T. Murray & Marilyn J. Murray.
p. cm.
Includes index.
ISBN 0-07-912945-5 (p)
1. Software maintenance. 2. Year 2000 date conversion (Computer systems) I. Murray, Marilyn J. II. Title.
QA76.76.S64M87 1996
005.1'6—dc20 96-649
CIP

Copyright © 1996 by The McGraw-Hill Companies. Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

4 5 6 7 8 9 0 DOC/DOC 9 0 0 9 8 7

ISBN 0-07-912945-5

The sponsoring editor for this book was Jennifer Holt DiGiovanna, the book editor was Kellie Hagan, and the executive editor was Robert E. Ostrander. The production supervisor was Katherine G. Brown.

Printed and bound by R. R. Donnelley & Sons, Crawfordsville, Indiana.

McGraw-Hill books are available at special quantity discounts to use as premiums and sales promotions; or for use in corporate training programs. For more information, please write to the Director of Special Sales, McGraw-Hill, 11 West 19th Street, New York, NY 10011. Or contact your local bookstore.

Product or brand names used in this book may be trade names or trademarks. Where we believe that there may be proprietary claims to such trade names or trademarks, the name has been used with an initial capital or it has been capitalized in the style used by the name claimant. Regardless of the capitalization used, all such names have been used in an editorial manner without any intent to convey endorsement of or other affiliation with the name claimant. Neither the author nor the publisher intends to express any judgment as to the validity or legal status of any such proprietary claims.

Information contained in this work has been obtained by McGraw-Hill from sources believed to be reliable. However, neither McGraw-Hill nor its authors guarantee the accuracy or completeness of any information published herein and neither McGraw-Hill nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that McGraw-Hill and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

In order to receive additional information on these or any other McGraw-Hill titles, in the United States please call 1-800-822-8158. In other countries, contact your local McGraw-Hill representative.

MH96
9129455

Chapter

11

The Conversion Plan

Most data-processing professionals are familiar with conversions that involve migrating to new hardware or new operating systems. User management generally understands the objectives of the change and looks forward to the benefits, fully aware that, based on past experience, the project will probably come in late and over budget.

Typically, a conversion requires changing the source code to accommodate syntax variations, transferring files to managed databases, and concurrently modifying the JCL. None of these processes is simple and none is anticipated with equanimity. However, all such conversions are routine if vendors have a stake in their success and, therefore, have at the very least committed resources to check-pointing the conversion path. To this extent, computer users haven't really been alone in their conversion efforts.

Probably the most extensive conversion efforts to date in the short history of computing are marked by the introduction of System/360 and the much later adoption of client/server architecture. Although these two events are separated by nearly 30 years, some installations are still executing 1400-series programs in emulation and are encumbered with awkwardly designed DASD equivalents of prehistoric tabulating systems—subtle testimony to the fact that IS personnel do not greet conversions with unbridled enthusiasm. It has been humorously claimed that IS managers have but a limited number of conversions in their constitution.

The current crisis requires a different kind of conversion, and doesn't allow for the luxury of flexible scheduling. Worse yet, it's worldwide in scope, mandatory in nature, consuming of resources, and totally without benefit other than saving us from a fate until now we didn't recognize. A successful conversion merely assures that we can continue to transact business beyond December 31, 1999, and for some the deadline is much earlier. User management is clearly not impressed by the return on investment.

There has been much procrastination and denial since we last addressed this problem in *Computers in Crisis: How to Avert the Coming Worldwide Computer Systems Collapse* by Petrocelli Books, Inc., 1984. A most important resource has been wasted in the interim—*time*.

The conversion we're contemplating now, more than 10 years later, must still analyze and alter stored data, re-create source logic, modify input protocols, and forever change programming practices, but all in a much reduced time frame. There are problems associated with client/server networks that will challenge the most creative technicians. Be assured that, due to the sheer force of numbers, we all must act alone. What vendor can muster sufficient resources in the face of such massive need? As was said earlier, "this is a user problem."

All of this explodes in the midst of a world economy totally dependent on computer resources for its survival and demanding of the services of skilled technical personnel whose availability is in dreadfully short supply. Yet we must succeed because there's no time to waste. This is a crisis so much of our own making that to hesitate to act immediately is irresponsible and inimical to national welfare, if not simple economic suicide. Nonetheless, we know there are those who will fail and we might very well be numbered among them; all we need to do is procrastinate.

Preconversion Objectives: An Overview

Generally, a conversion is an exceptional activity to be completed one way or another and gotten out of the way of the application backlog so the important business can go forward. Consequently, conversion techniques occupy a minor place in computer science and technology. Unfortunately, you'd be hard pressed to find mention of conversion techniques in the literature, other than in project control references—an area not addressed by this book.

Lengthy research uncovers a single book on the topic: *Conversion of Computer Software* by John R. Wolberg of the Technion-Israel Institute of Technology. Turning to a review of IBM resources on this subject, you'll discover GC28-1251-00, *The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation*. A dearth of material, to be sure, but at least a beginning.

Observing the two offerings, one eminently academic and the other eminently practical, we're brought face to face with the chasm between computer science and data-processing practice. One approach, computer science, is seemingly preoccupied with conceptualizing while the other, data-processing practice, is obviously preoccupied with getting the job done. Our task is clearly one of combining the two.

Conversion is a three-stage effort comprising preconversion activities, implementation, and postconversion activities. The preconversion steps produce parameters that reveal the apparent implementation resource requirements: machine time, staffing needs, and duration. The implementation stage addresses the active application of the solution. The postimplementation stage addresses cleaning up, monitoring, and observing the installed system. Preconversion steps are best kept separate of requested system modifications, enhancements, or addenda. Preconversion activities target date usage and provide the parameters necessary to schedule the conversion's implementation. When later considered in relation to the application backlog, these

resulting parameters should aid in identifying backlog projects that will be deferred until the completion of the implementation stage. Undertaking extraneous technical efforts, which only adds to the inventory of problem programs even as we work to solve such problems, doesn't make sense.

The steps that must be taken in the preconversion stage are explicit. They are a product of the available literature, extensive consulting experience, and an intimate awareness of the social and political pressures brought to bear in both large and small computer user environments. Above all, the preconversion stage must produce knowledge, and seeks to answer the following six questions:

- What are IS's technical options?
- How many work-days are implied by the program and database inventory in need of repair?
- What machine resources are required for this conversion?
- Given the required versus the available resources, what's the time requirement to complete the implementation?
- Since continued backlog programming and system development will add to the conversion requirements, can activity be suspended until implementation is completed?
- What's the total cost?

Technical Options

The preconversion steps depend on assumptions about the status of the system. Of primary importance is the status of the system date. Will the accessible system date be six digits or eight digits? A good deal hinges on the answer to this question. Only one vendor, IBM, has issued a statement of intent to provide eight-digit dating for its software current with the end of 1996. If only a six-digit date is accessible, an early problem develops in that TIMESUB2 creates the Neojulian date from an eight-digit input date.

The century table

In order to access an eight-digit date when there's only a six-digit date available, it's necessary to invoke a century table, also known as a "sliding-century window." You must decide how many years in the past and how many years into the future the table will function. This is governed by the firm's computing horizon. Generally, the current year minus 35 years and plus 64 years define the century table's architecture, although any contiguous 100-year span is allowable. Thus, assuming a current year of 1995, you're looking at a table whose lowest entry is 1960 and whose highest entry is 2059. Here we define 19 and 20 as the table functions while 60, 61, 62, . . . 00, . . . 59 are the table arguments. The search argument, of course, is the two-digit year component of the six-digit date whose century you're attempting to isolate.

If the table is programmed so that each year it increases all of its components by 1, the century table remains in relative step with the calendar. The benefits of the table

are obvious in that a six-digit date can be input, its two-digit year used as a search argument in the century table, and the two-digit century table function returned. Six-digit year-2000 dates merely execute the table lookup with the two-digit year 00. When a match is found for the table argument, 00, the table function is accessed, 20. Thus the four-digit year 2000 is assembled and made available to the program.

All of this works, however, only if no date prior to 1960 enters the system. The century table is an excellent emergency solution for firms operating well within the limits of the table's effectiveness, and with no foreseeable future exposure to merger, expansion, or accelerated growth. Unfortunately, all too often a problem with using the century table is the tendency to standardize it. Advocates' typical claims often sound partisan:

"Now that we can acquire a four-digit year from the century table, we can invoke any of the subroutines we have available to accomplish the program corrections required. Quite simply, it's possible to create a program environment whose logic is internally eight-digit-date-oriented while all of its output is six-digit dates. Thus, we can patch programs without having to update databases. This approach reduces the total workload in that the stored data is untouched. Even more attractive is the fact that individual programs can be updated and returned to service after testing—there's no need to go through the rigors of lengthy planning and subsystem scheduling."

Beware! For the effort that's saved versus the risk taken, we advise the more thorough approach. Conversion to an eight-digit-date standard both within programs and within databases staves off ultimate disaster and preserves the investment demanded by the current crisis. While the options must be explained to user management, so must their hazards.

Among the aftershocks of our present crisis will be a clearer view of IS as either a financial asset or a liability. Typically, IS departments are rarely audited prior to a merger or acquisition. After corporations are confronted with multimillion-dollar invoices for their share of the current cleanup effort, however, they'll very likely be extremely circumspect about a possible future liability for what could be called a "less than thorough" date conversion. Future mergers and acquisitions might well hang in the balance.

IBM offerings

Various vendors offer technical options. IBM has developed a COBOL compiler for MVS and VM. Another compiler, VS COBOL II, is available as an intermediate migration aid for those using the discontinued OS/VS COBOL compiler.

VS COBOL II doesn't provide sliding-century window support or intrinsic functions, although, as we mentioned earlier, IBM has announced its intent that the most recent releases of IBM software products as of year-end 1996 will support dates and date fields for A.D. 2000 and beyond.

COBOL for MVS and VM features ANSI COBOL standard intrinsic functions and a sliding-century window. A multilanguage runtime library and language environment for MVS and VM supports PL/I for MVS and VM, COBOL for MVS and VM, C/C++, Fortran, and MVS/ESA, and also provides additional date manipulation support.

PL/I for MVS and VM, like COBOL for MVS and VM, brings with it four-digit-year support, sliding-century window support, and built-in functions—PL/I's equivalent of the intrinsic functions of COBOL.

The VSE environment offers COBOL for VSE/ESA, a compiler, and language environment for VSE/ESA, the runtime library. ANSI COBOL standard intrinsic functions provide four-digit-year date manipulation capability. Sliding-century window support is also available; it isn't an ANSI facility. Also available to the VSE market are COBOL compilers DOS/VS COBOL and VS COBOL II, neither of which can handle four-digit-year dates. VS COBOL II is recommended as an intermediate compiler to the installation of COBOL for VSE/ESA.

PL/I for VSE/ESA, a compiler, and language environment for VSE/ESA, the runtime library, provide a sliding-century window and built-in functions.

There are three RPG compilers, focused mostly on the AS/400: the Integrated Language Environment (ILE) RPG IV, Original Program Model (OPM) RPG/400, and a System/36-compatible compiler. The first two compilers, ILE and OPM, allow you to retrieve a four-digit-year using *YEAR or *DATE. The two-digit date components or the full six-digit date is subject to retrieval via UDATE, UYEAR, UMONTH, and UDAY—analogs to *DATE, *YEAR, *MO, and *DAY. The System/36-compatible compiler is restricted to the two-digit year of the six-digit-date environment.

Built-in calendar functions and intrinsic functions

Collectively, additional functions featured in a compiler or language environment (runtime library) provide for the accessibility of a four-digit-year as well as limited date arithmetic. Intrinsic functions return values. In one intrinsic function for VSE COBOL, submitting DAY-OF-INTEGER returns YYYYDDD, the neo-Julian date, while submitting DATE-OF-INTEGER returns YYYYMMDD, the full eight-digit date.

RPG arithmetic facilities for the AS/400, for instance, allow duration calculations; ADDUR (add duration) adds years, months, or days to a given date. Similarly, subtraction can be performed. The results, however, are prone to the same vagaries encountered in extended-interval aging.

We could go on to review the feature offerings of any number of vendors after exhausting IBM's inventory of software for the year 2000. There are a multitude of CASE tools designed to aid re-engineering and tools to unlock the secrets of files. Space and time, however, don't permit an in-depth review.

We earlier urged that this conversion would be well served if other agendas were suppressed. Indeed, this is true. Nonetheless, we see that upgrade software can be purchased in the midst of this, the world's most widespread conversion effort. Computer users must determine for themselves the feasibility of attempting to upgrade to a more contemporary language compiler or operating system while solving the year 2000 date problem. Many installations have become seriously out of date and could have much work to do just to be able to attempt this conversion.

Sizing the Task

An informed judgment can't be rendered without numbers. The IS quest for knowledge addresses the number of programs in which date calculations are taking place. In stored data (databases), where are dates, date derivatives (date elements used as counters, and so on) or dates used in file identification or indexing? Where are dates entering the system? How are they being used? Are all dates equal in importance?

By isolating an instance of date input, you might be able to see the relative unimportance of some dates. Consider the following electronic data interchange (EDI) scenario:

You receive an EDI purchase order containing a six-digit date that your system ignores in favor of its system date as the order date. Other suppliers receiving this EDI purchase order might use the date as a sort field. Due to this diversity, it's difficult to determine truly important dates. It's incumbent on IS to make its system date proof.

A case in point

Perhaps the most delicate part of sizing the task is making determinations relative to a given date's true activity. While it's necessary to closely scrutinize each detail, it would be naïve today to think that you could size the conversion task efficiently without the aid of software tools.

Nonetheless, several years ago we undertook the chore of sizing an anonymous client's application program, utility, and database inventory. To determine the degree of difficulty and the probability that the task could be automated, we performed the sizing manually. It required two people working for 96 hours. The findings were as follows: All database records bore dates. Fifty-two percent of application programming was obviously involved in date usage. Sort utilities constituted approximately 15 percent of the involved software.

Reviewing the installation's change management documents permitted us to peek into the productivity of the programming personnel. The programming backlog was reviewed and prioritized, resolving conflicts with the programs in need of date format correction for the year 2000. Invoking the installation's programmer productivity expectations, we concluded that the staff had to be increased by 30 percent in order to complete its task by 1999. This client task was to be handled totally in-house using manual techniques.

Given the time frame, number of programs, databases, utilities, complexity of the task, and lack of telecommunications involvement, early resolution was promising. Today's analytic software tools would reduce the size of the effort considerably, but there will always be the difficulty of determining the importance of some dates. No tool is a panacea.

The end of this chapter has a list of firms that have identified themselves or their products as being useful in one or another area of the conversion effort. We haven't reviewed their products or services due to time constraints, but have listed them for your convenience. Send one or more of the companies a query letter or postcard for product information. We've omitted telephone numbers for the sake of the vendors and potential clients who could end up on hold.

Machine Resources

Machine time is always in short supply and, as more complexity develops, computer availability becomes a key factor. While the task-sizing effort is going forward, the installation's total application system must be separated into a group of noninterlock-

ing subsystems. The smaller these subsystems can be made, the less complex will be the implementation of the solution plan. In cases where dependencies are permanent, additional ancillary programming must be created to free the subsystems.

The sizing task addresses not only the location, type and, quantity of date usage in programming, databases, utilities, JCL or CL, and screens (CICS, or display files) but added to this are client/server and miscellaneous networks, as well as free-standing PCs. Because the days of single-vendor procurement have disappeared, these devices and their software could pose an enhanced learning curve at the operating system level where dates dwell.

Machine resource requirements can be assessed with software that estimates run times based on subsystem performance. You can also find programmers' machine requirements reflected in the records of the change management logs. In the event that machine availability is extremely tight, the only solution might be to employ an additional computer configuration.

Because the modified subsystems will be tested at the unit level then at the subsystem level, and finally executed in parallel with the production subsystem, an additional computer configuration might be mandatory. It isn't a luxury to parallel both the test system and the production system when modifications are complete; machine time requirements might continue to grow.

If you'll be conducting testing on the installation's production machine, take considerable care to ensure that changes to the system's date are isolated. Several of the vendors listed at the end of this chapter offer date-change software.

This conversion is pervasive. Other conversions have addressed specific differences; this one will take you where you don't want to be. Be aware that your month-end, quarter-end, and year-end closing programming could be affected by date modifications. It might be necessary to execute a system test using month-end, quarter-end, or year-end data from a prior period. And you'll need machine time to test areas that other conversions won't affect.

The Time Requirement

Your ability to arrive at a total implementation time requirement will be no better than the quality of your installed change management system. Reviewing changes made to the system allows you to become familiar with the installation's level of productivity. Using Wolberg's productivity equations offers some help, but the short time available to solve this problem leads to some more intuitive measures.

The total time requirement consists of estimates. The total number of lines of code requiring modification divided by the installation's programmer productivity level constitutes the rock-bottom work-day estimate. If the decision has been made to use a century table, only this programming can be involved.

If you decide to execute a proper conversion, you must add to the total time the work days required to create the "throw-away" interim programs to isolate the subsystems into separate entities. Presumably, the whereabouts of dates in the databases has already been roughly determined and the requirement to create database updating programs evaluated. You must also add the work days required for these procedures.

At this point, total the work days. It's now time to begin the resources-versus-work-days evaluation. Few organizations carry surplus personnel on their programming staff. The solution to the date problem is, in all likelihood, going to require additional personnel. New personnel result in the expenditure of location and hiring time as well as time for the new employees to become familiar with the installation and its programming practices. This time must also be added to the implementation time total. At long last, total the full time requirement. Few will be happy with the results.

The longer an installation procrastinates, the less likely it will be able to "staff up" with capable personnel. The marketplace will become more and more competitive as the pool of professional personnel dries up. Companies might need to make counter offers to disaffected personnel bent on resigning for opportunities elsewhere. And beyond all of this is the possibility that the implementation stage will overrun the time calculated, placing additional pressure on the IS team.

Suspend backlog development?

It's important to review backlog tasks, which tend to execute under the status quo, thus adding to the conversion problem. Quite often there's an opportunity to place backlog projects in limbo until the implementation stage has ended. Having presented the implementation time resource requirements, management is often quite receptive to any possible reductions.

The benefits of a backlog furlough are valuable. Personnel not working on backlog tasks are available to participate in conversion implementation, and these work days can be subtracted from the total. Further, fewer active backlog tasks mean less competition for resources.

Clearly, no one benefits from backlog programming that's written one day and discarded the next. Backlog expansion need not be stopped, but programming in this area should be restricted if not put on hold.

What Does the Conversion Cost?

By no means will the conversion into the land of eight-digit dates be inexpensive. Costs of thousands, tens of thousands, hundreds of thousands, millions, and even tens of millions of dollars are commonplace. Those who have attempted to calculate the cost of solving this problem on a worldwide basis have ended up with estimates of more than 300 billion dollars. As would be expected, the more dependent an organization is on computer support, the more likely its program library and costs will be huge.

Certainly, after executing the preconversion steps, the dollar amounts should be known; they're the product of the work days multiplied by the cost of personnel, plus software tools, plus the cost of lost profit opportunities. Unfortunately, profit opportunities are also lost to more aggressive competitors who have already converted.

If the final dollar count is the determining factor in pursuing a solution to the date problem, you're placing a very low value on your firm's business. We've also possibly missed something in our analysis and presentation.

Consider what will happen if management turns down the project after the preconversion effort. The presence of formal management controls, accountability, and

formal reporting often make the difference. A pilot implementation might also aid in tying down elusive times and cost figures.

Management Protocols

Step 1

In preparing for the preconversion effort, you should select an in-house consultant from among a technician pool. Once the preconversion phase starts, he or she should report to the IS director and to the chief executive officer jointly. The in-house consultant is charged with the task of answering the six questions outlined earlier in the chapter.

The consultant's responsibility to IS is technical, and his responsibility to the CEO is informative. This triumvirate defines the preconversion management team. If staffing levels are so low that a consultant-caliber technician doesn't exist or can't be spared, or if the fortunes of business preclude the luxury of appointing a consultant, the effort is at mortal risk.

Remember that this crisis will destroy many firms that have waited too long to begin their conversion effort. Yours could be one of these, and you'll know this early on.

Step 2

Having been assigned this task, the consultant must become familiar with the nature of the problem of date computation by thoroughly reading the contents of this book. Additionally, the consultant should review IBM's earlier-cited GC28-1251-00 and, if possible, Wolberg's work on software conversion.

Step 3

The consultant, already familiar with the installation's system structure, identifies, in concert with IS, the technical resources required to carry out the steps of the preconversion stage. This step culminates in a joint meeting with the CEO. The meeting's agenda is the first preconversion document. The objective of this meeting is to enumerate and procure the required resources for the successful completion of the preconversion task. The CEO should, in fact, be asked for a permanent, firm commitment of those resources.

Just as there's nothing informal about asking for corporate funds, it's not politically wise to allow room for failure by compromising the task's true requirements. It's important to formally prepare a meeting agenda containing the enumerated resource requirements. It's equally important to prepare a postmeeting memo that recapitulates the meeting's conclusions.

Among the resources often requested are additional personnel to support the preconversion activities, purchase software to reorganize "spaghetti code" COBOL programs, develop program file/record/field usage documents, identify date fields within the database, and so on.

The remaining steps define the work to which those resources must be committed without reservation. The CEO must be apprised of the reason for the request of each resource.

Step 4

The targets of the conversion are: application programs, sorts, utilities, JCL, source library inclusion books, card decks in drawers, data entry formats (desktop terminals, key to tape, key to disk, OCR, data communication, punched paper tape, bar code, plastic card, diskette, hand-held terminal, tape cassette, prepunched style tag and POS are but a few), and the database. Consequently, an inventory must be made in preparation for a subsequent meeting with IS and the CEO.

The objective of this inventory is to isolate mutually exclusive, date-dependent subsystems. Mutually exclusive subsystems are discrete groups of programs, sorts, or utilities that involve computation of dates but share no common files or other common data resources. The goal is to create subsystems whose date-related program logic and files can be converted and returned to production to function in an eight-digit date environment independently of other systems. The corresponding data-entry formats are necessary to assure conversion of the input interface. The related JCL is required for subsystem testing and debugging as well as verification of file identification. These objectives must be explained to the CEO.

It's necessary to review the installation's entire application software collection in order to compile this inventory. The person or people conducting this review must scour operations schedules, special request procedures, and library directory listings. Perhaps the most disheartening discovery in this process is uncovering executing phases for which no source books are available or for which the synchronization between source and object no longer exists.

The degree to which an installation is organized will determine the degree of difficulty encountered in compiling an inventory. If no installation-wide program/file cross reference or data dictionary exists, it will be necessary to consult JCL (production cataloged procedures) to identify the data resources. We offer the following guidelines:

- If your installation doesn't now employ a maintenance and change control system, one must be implemented. The spirit of such a system is that no programming or design be undertaken without the receipt of a written assignment from IS management. Thus, a formal backlog is maintained, complete with time estimates. The consequence of change control is that the consultant can be kept continuously aware of the status of the elements he or she is attempting to inventory. This system will be of even greater importance during the implementation stage.
- Compile a list of files containing date fields, parts of dates, or time-related parameters, for example day, month, or year counts. You might want to divide your file collection into categories according to media: cards, diskettes, screen, telecommunication transmission, and so on. The important consideration is that this list must establish a complete enumeration of date-related files. (Of equal importance with permanent files are those intermediate files and work files that have brief lives, often created and destroyed within the same job.)
- Now it's necessary to review the installation's entire programming collection. The objective of this review is to determine the programs, sorts, and utilities that address the data resources that were identified in the previous guideline. A file/pro-

gram cross reference chart must be created and, if at all possible, segregated into mutually exclusive subsystems.

Step 5

The output of Step 4 combined with the peculiarities of your installation's fiscal closing, industry peak periods, and applications development in progress allows you to establish a tentative conversion sequence.

Recognizing that both maintenance and change/enhancement must be frozen for the program components of each subsystem during the conversion, the consultant must consider multiple factors in the final prioritization of components within subsystems. Further, if multiple small subsystems were not developed in Step 4, the conversion implementation team requirements will be greater in order to reduce the duration of these freeze periods. Thus, prioritizing the component programs is to some extent governed by the programs' change and maintenance volatility.

Step 6

Now you must detail the conversion methods and procedures and the standards that must be maintained during the implementation. Here we're referring to such things as the form and extent of a century table if this has been chosen as a integral part of the conversion. If routines from this publication are to be used, will they be macros or subroutines? If they're necessary, are the linkage conventions available?

Have subsystem conversion teams been tentatively designated? No programmer must be left to his or her own devices in implementing any date computation solution. The more comprehensive the developed details of the implementation, the more likely management will be inclined to give it the go-ahead.

Finally, there's the cost and necessity of creating documentation that reflects the changes made to the system. This encompasses users' manuals as well as programming documentation and system documents.

Step 7

A set of programs and related files must now be chosen for the purpose of executing a pilot conversion. It is from the pilot conversion that specific time utilization parameters are collected for comparison with estimates made earlier. Consequently, the sample must be chosen by the consultant with considerable care, being aware that it will test the recently established procedures and standards as well.

While the inventory was being compiled in Step 4 or while the standards were being created in Step 6, various levels of conversion complexity were recognized for each programming conversion problem. It is important that the pilot conversion sample contain a representative assortment of these complexities.

It should be noted that only the more common problems have been addressed by the subroutines presented in this text. Two-digit year-numbers will be encountered in peripheral storage, allowing their replacement only by binary equivalents of the full four-digit value if the same storage bytes are to be occupied, thus avoiding reformatting. The unexpected becomes the expected given the performance of programmers over the past 50 years.

Step 8

Having successfully completed the preponderance of the preconversion steps, it is now necessary to create a document that will embody the planning to date. The technical solutions, aids, ancillary programs specified, the anticipated staffing and team structure, as well as the assignment of responsibilities and the schedule for the implementation of the parallel testing must each be clearly stated. This document is the repository of the entire effort to this point. The sequencing of the conversion, the training requirement for the participants, the programming standards, and the testing protocols are all within its content. The benefits are twofold:

- An objective record is created with which the members of the preconversion team may reach agreement.
- A basis is provided for modification and improvement upon the acquisition of practical experience.

Because parameters are to be sought vigorously during the execution of the pilot conversion, it is important to lay out carefully individual work plans that will facilitate the comparison of estimated to actual performance. Among other benefits, the pilot conversion allows the testing of the change/maintenance control system as well as the procedure for logging needed modifications to the procedures themselves as they become apparent. To facilitate all of this, it is important to build a reporting facility into the pilot conversion's work planning.

Step 9

Execute the pilot conversion. On the assumption that the technical homework has been done well, the emphasis shifts from the technical to the managerial as the preconversion activities become involved with the pilot conversion now underway. The importance of an adequate project control system cannot be overstated at this stage.

It is during this pilot conversion that the team participants will be trained, the ancillary programming tools will be created, the programs in conversion "frozen" with respect to further maintenance or change, and in every respect a mini-implementation executed. While we are interested in comparing actual *versus* estimated parameters for each aspect of the pilot conversion, it is the collection of the actual parameters that must occupy our primary focus.

A Time for Decision

Having completed the pilot conversion and having acquired the much needed parameters, it is now possible to perform the calculations that will allow human, machine, and time resource requirements to be associated with the entire inventory. If a mathematical result is sought, Wolberg's work contains resource requirement calculation formulae that may be useful. The process of extending the parameters gathered during the pilot conversion is largely based upon common sense and simple arithmetic.

The consultant is now able to answer the six questions initially presented:

- What are IS's technical options?
- How many work days are implied by the program and database inventory in need of repair?
- What machine utilization resources are required for this conversion?
- Given the required versus the available resources, what is the time requirement to complete the implementation?
- Since continued backlog programming and system development will add to the conversion requirements, what are our options in suspending activity until implementation is completed?
- What is the total cost?

A final meeting between the consultant, IS director, and the CEO now permits a complete presentation of the forthcoming conversion plan. The total cost of the date conversion may be calculated and the final implementation and postimplementation activities scheduled.

What of the preconversion stage? Are there alternatives to employing an in-house effort?

We have just reviewed an intuitive, and a more formal approach to an in-house conversion. Most conversions will fall somewhere between these poles. There is the possibility that management may want to outsource its conversion hoping for a fixed dollar cost for the whole effort.

IBM's Information Systems Solution Corporation (ISSC) offers consulting services as well as an "Assessment and Strategy" session that covers an eight- to twelve-week period during which ISSC offers to identify the magnitude of the client's date problem.

ISSC offers to direct this effort to clients executing 5 to 150+ million lines of code in a mixed language environment. The user might view these eight- to twelve-week parameters as minimums given the case that ISSC personnel are specially equipped to perform this service.

The preconversion stage is of paramount importance. It provides the basis for intelligent judgment regarding the installation's future course in the face of the relentless march of time . . . a time for decision.

Vendor Tools and Service Offerings

ADPAC Corp.
425 Market St.
San Francisco, CA 94105

COGNICASE
425 Viger Ouest
Montreal, Quebec H2Z 1X2
Canada

Computer Software Corp.
19100 Detroit Road
Cleveland, OH 44116

Compuware Corp.
31440 Northwestern Hgwy.
Farmington Hills, MI 48334

EDGE Information Group
2250 East Devon
Des Plaines, IL 60018

Forecross Corporation
90 New Montgomery St.
San Francisco, CA 94105

Global Software, Inc.
P.O. Box 2813, 15 Depot St.
Duxbury, MA 02331

Ironsoft, Inc.
4323 Winnequah Rd.
Monona, WI 53716

ISOGON Corp.
330 Seventh Ave.
New York, NY 10001

Izar Associates, Inc.
4 Emery Ave.
Randolph, NJ 07869

Mainware, Inc.
7176 Pioneer Creek Rd.
Maple Plain, MN 55359

Micro Focus
2465 East Bayshore Rd.
Palo Alto, CA 94303

Quintic Systems, Inc.
3166 Des Plaines Ave.
Des Plaines, IL 60018

Reasoning Systems
3260 Hillview Ave.
Palo Alto, CA 94304

REPPIN Consulting and
Software Ltd.
P.O. Box 6912, Station J
Ottawa, Ontario K2A 3Z5
Canada

SEEC, Inc.
5001 Baum Blvd.
Pittsburgh, PA 15213

Software Eclectics, Inc.
10955 Jones Bridge Rd.
Alpharetta, GA 30202

TransCentury Data Systems
111 Pine St.
San Francisco, CA 94111

VIASOFT
3033 North 44th St.
Phoenix, AZ 85018

McGraw-Hill Professional Publishing

Search Catalog/Order

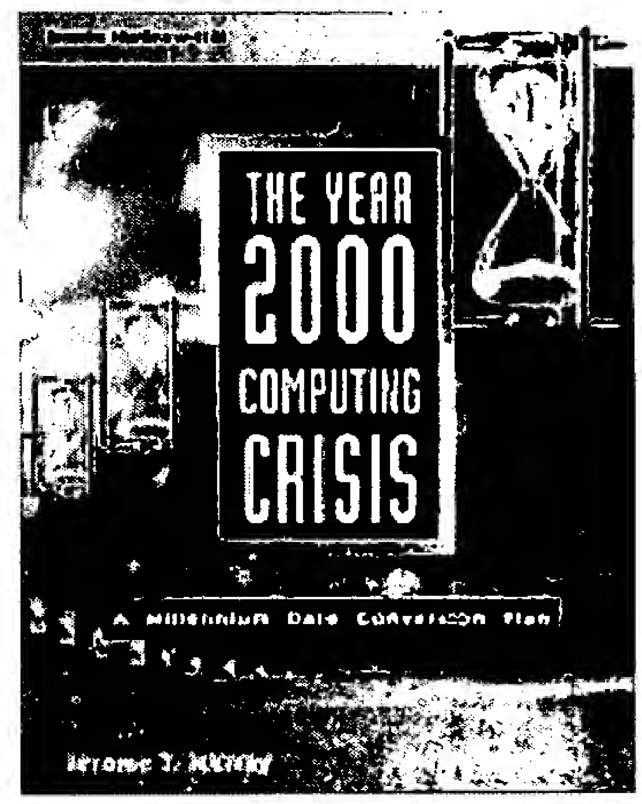
- PPG Home
- Contact Us
- Customer Service
- For Authors
- International Offices
- New Book Alert
- Search Catalog/Order
- Site Map
- What's New

- Bet@ Books
- Business
- Computing
- Education & Training
- Encyclopedias & Dictionaries
- Engineering
- General & Recreation
- Healthcare & Medicine
- McGraw-Hill OnLine Learning
- Online & Electronic Products

The Year 2000 Computing Crisis: A Millennium Date Conversion Plan

Jerome T. Murray
Marilyn J. Murray

Standalone software: 0-07-912945-5
(Mar 1996)
$39.95US
321p. 44 Illus. 7 3/8 x 9 1/4



The year 2000 computer crisis--solved! Scrambling to find a solution to the date conversion dilemma your computer system faces as the 21st century approaches? Bracing yourself for the potential ``bad data" nightmare? Relax, Jerome and Marilyn Murray's The Year 2000 Computing Crisis: A Millennium Date Conversion Plan has the answers. This urgently needed rescue guide maps out systematic strategies for sidestepping virtually every problem you face--from minor software glitches to complex telecommunications foul-ups to macro installation to tal system collapse. Exhaustively tested and retested, it delivers a comprehensive conversion plan that includes a source code disk of solutions written in COBOL, RPG and assembly lanuage--giving you maximum flexibility and convenience.

** Review **

"Useful as a collateral text for computer systems engineering classes or agency-run programs engaged in date conversion."--Science Books & Films, 12/96

** Contents **

Time Measurement: A Brief Review. Algorithms, Notation, Pseudocode, and Programming. The 8-Digit Date Puzzle. Short Interval Aging. Extended Interval Aging. The Translation Algorithm. Data Versus Name of Day. Conversion of the Neojulian Date. The Algorithm of Easter. The Gregorian Status Indicator. The

Conversion Plan.

Subject(s): Computing--General and Reference

PURCHASE OPTIONS

1. Order by phone **Individual U.S. Customers:** Call 1-800-2-MCGRAW (1-800-262-4729) 24 hours/day - 7 days/week **U.S. Federal Government Customers:** Call 1-888-878-5150 Toll Free 24 hours/day - 7 days/week **Bulk Purchases:** Call 1-800-842-3075 8:30 AM - 4:30 PM ET **International Customers:** Visit our International Offices page to find a McGraw-Hill office near you.	**2. Visit Your Local Bookstore** **3. Order Online Right Now Using Our Secure Order Form** **4. Order via mail or fax (Individual customers only)** Click here for order form and instructions.

Professional Publishing Home | Contact Us | Customer Service | For Authors | International Offices | New Book Alert | Search Catalog/Order | Site Map | What's New

Bet@ Books | Business | Computing | Education & Training | Encyclopedias & Dictionaries | Engineering & Science | General & Recreation | Healthcare & Medicine | McGraw-Hill OnLine Learning | Online & Electronic Products

McGraw-Hill

A Division of The McGraw-Hill Companies

Copyright © 1998 The McGraw-Hill Companies. All rights reserved. Any use is subject to the Terms of Use; the corporation also has a comprehensive Privacy Policy governing information we may collect from our customers.

ACM Digital Library

International Conference on APL

Conference proceedings on APL 90: for the future
August 13 - 17, 1990, Helsingor Denmark

What's ahead for 2000 A.D.?
Page 364

Howard J. Smith

metadata: abstract index terms
full text: PDF 73 KB
[Find Related Articles | Add to Binder]

ABSTRACT

On January 1, 2000 many applications will stop working properly. Most utilities and applications express the year as the two low order digits of the full representation of the year

What's Ahead for 2000 A.D.?

Howard J. Smith, Jr.
IBM Corporation
1501 California Avenue
Palo Alto, CA 94304

On January 1, 2000 many applications will stop working properly. Most utilities and applications express the year as the two low order digits of the full representation of the year so that dates in the new century will appear to occur before dates in the old. When this phenomenon occurred in the past humans adapted quickly, and no great dislocation occurred.

Unfortunately, the computer is not as adaptable or reasonable as the human.

This paper presents one possible solution to the problem, at least in the APL environment. This solution involves replacing date fields in files and records with a pseudo-Julian day number (PJD). This number permits the analyst to use date information arithmetically to determine the day of the week, the period between dates, create lists of dates, etc. The PJD is easily converted to or from human-usable canonical date forms. Finally, the technique defers difficulties due to changes of year, century, millennium, etc. indefinitely.

In the later sections of the paper some thoughts on possible extensions to the leap-year algorithm are discussed. The new terms would improve precision without affecting the past or present. Finally, sample functions which support and utilize this "perpetual" calendar are given.

It must be emphasized that this paper represents a solution only within the APL environment. The concept is applicable in any environment, language or system but only if it or a similar solution is adopted before December 31, 1999. If we users of APL recognize and fix the problem in our environment while the rest of the computing world ignores it, the problem will still affect everyone, including us.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1990 ACM 089791-371-x/90/0008/0364...$1.50

EDS

November 15, 1991

William S. Lawson
Administrator for Documentation
US Patent and Trademark Office
Washington, DC 20231

Dear Bill:

We have run some additional analysis since our last follow-up report of October 11. This particular analysis was specifically oriented toward the suggestion that for your very large database, initial classification by machine should be equivalent to that by personnel being trained. Consequently, the analysis that we ran is called a Jackknife analysis in statistical terms. To illustrate, it is like folding each observation in turn back into the set itself.

That is exactly what we did. Each patent was classified against the background of the remaining 7,209 patents that would have existed in the database. With this we obtain a more accurate measure. As I indicated in our meeting, it is something less than the 98.5% that we were able to achieve using Bayes 3, and it is a little better than the diagonal values of the initial USPTO classification of 77.4%. It is, in fact, 82.3%.

Based upon this very large and well considered sample of 7,210 patents, it appears that if a new patent application within the field that is covered by them were considered by the automated methodology, it would place the application in the proper category 82.3% of the time. In addition, for those that do not get placed directly on the diagonal you would receive distance measures or at least the ordering of potential categories. Thus, it would be easy for the Patent Examiner to select from among the top 3, 4 or 5 choices, whatever your office should decide.

Research and Advanced Development
7223 Forest Lane
Dallas, Texas 75230

Therefore, in response to Ken Dood's question about first time accuracy, we feel that this experiment provides the best answer we have to that particular point. I trust that you will find this useful. If there are any other questions that your office might have, please feel free to contact us.

Sincerely,

Glen Self
Vice President, R&D

Attachment

cc: Ken Dood
Becky Rudolph

JACKKNIFE ANALYSIS OF BAYES 3

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	RECALL
0 - 0	(433)	**356**		1	6		3	7	1	4	4		1	7	7	8		3	3	22	82
1 - 1	(254)		**186**	3	20	2	4	2	4	3	10				2	17				1	73
2- 2	(268)	6	3	**178**	12			11	3	3	1		4	21	4	10		2		10	66
3 - 3	(616)	12	18	1	**505**	12		12	2		1			5	3	41		1		3	82
4 - 4	(253)		1		14	**197**		7	6	3	8	1	1	1	6	7			1		78
5 - 5	(138)	5	2	3	2		**71**	3	2	5	4	4	1	2	9	18	1	2	1	3	51
6 - 6	(318)	5			8	4	3	**267**	4	1	4	1		3	2	4			1	11	84
7 - 7	(274)	2	8		13		5	15	**160**	7	4		1	16	7	21		3	7	5	58
8 - 8	(273)		5	3			1		1	**233**	6	2	3	5	1	1	1	7		4	85
9 - 9	(240)	4	7	2	4	4	5	1	3	8	**181**	9	5		2	4				1	75
10-10	(102)			1			7		7	4	6	**63**	4		2	5		2		1	62
11-11	(372)	3	1	4	1		1	1			2	1	**315**	22	9			4		8	85
12-12	(691)	11	2	33	9	7	1	2	10	10	1	1	21	**542**	10	15		2	1	13	78
13-13	(731)	3					1		2	1	1	2	7	1	**694**	4				15	95
14-14	(589)	5	4	10	19	10	2	7	3	1	2		2	3	3	**507**	1	4	5	1	86
15-15	(422)		3				1	2		11	3	3		1		4	**383**	8	2	1	91
16-16	(840)	3		2			5	1		15	1	3		2		18	13	**773**		4	92
17-17	(91)							1	2							4			**84**		92
18-18	(305)	14		2	4		5	7	6	3	3		2	5	20	5		3	1	**225**	74
TOTALS:	7210	429	240	243	617	236	115	346	216	312	242	90	367	636	781	693	399	814	106	328	5920
PRECISION:		83	78	73	82	83	62	77	74	75	75	70	86	85	89	73	96	95	79	69	

EDS

September 19, 1991

Gerald Goldberg
U.S. Department of Commerce
Patent and Trademark Office
Washington, DC 20231

Dear Mr. Goldberg:

The attached report is self-explanatory. We feel comfortable with the results and feel that it demonstrates well the technology that we have put together. In retrospect, it is unfortunate that your office went to so much work to do the original detail classifications. We felt that a training set of approximately twenty per category would have been sufficient, and we could have reached this experimental point much sooner. Regardless of that, by virtue of your having done all the work and provided such an exhaustive training set, we were able to demonstrate very good results. More specifically, we were able to demonstrate the capabilities of the Bayesian network as a learning methodology. We hope that by virtue of our discovering 41 additional patents, you will receive some insight into our distance calculation and how we are able to perform it automatically.

I will be in the Washington area on the afternoon of the 30th and will contact your individual offices next week to determine if you would like to discuss anything further concerning this effort or if you are satisfied with the results. We sincerely hope that this is of assistance to the U.S. Patent Office. From the standpoint of the Patent Bar, I would like to see them charged with the knowledge which is made available through this type of technology.

The obvious application is for classification of the patent application. The down-side is that the application would need to be scanned. The good news is that you simply scan it in and use that part of the text which the scanner will accept to. Therefore, for purposes of classification and assignment to an examiner, you would not have to clean up the document or worry about bitmaps.

Research and Advanced Development
7223 Forest Lane
Dallas, Texas 75230

U.S. PATENT & TRADEMARK OFFICE

Memorandum

To: Group 2300 Examiners

From: Ginger Roberts
In-Group Search Assistant for 2300

cc: Alvin Oberley - 2316

Date: November 15, 1995

Re: APS Enhancement

With the Messenger Release S93.1/S93.2, the SELECT command is enhanced so that it can also be used to create, from specific display fields, a set of terms called a "term set." The attached document describes how to create, search, and display a term set. (Terms can be classes or even keywords.)

This enhancement enables you to:

- ✓ create an L-numbered set containing terms from patents obtained in an answer set without selecting e#s
- ✓ display a distribution of terms based on occurrence, number of documents or percentage counts

If you would like a demonstration of this latest enhancement or a copy of a search example, please stop by 4R20 or call me for an appointment.

Thank you.

2. Enhanced Select Command

The SELECT command is currently used to create an E-numbered list of terms from one or more patent display field(s), allowing you to search the individual terms using the corresponding E-numbers. With Messenger Release S93.1/S93.2, the SELECT command is enhanced so that it can also be used to create, from specific display fields, a set of terms called a "term set." Each term set created using the SELECT command is assigned an L-number. Once created, an L-numbered term set can be used to search the *entire* group of terms in the set connected by OR logic.

Creating a Term Set

To use the SELECT command to create a term set, you must first enter the following SET TERMSET command. Then enter a search of your choice in the file of your choice, as shown in the following example from the TRAIN file:

```
=> set termset L#

=> file train

=> s catalog
L1    23   catalog
```

Next enter **SELECT**, the L-number of the answer set containing the patents you want to select, the display format(s) for the patent section(s), and the number(s) of the answers. A maximum of 50,000 answers can be selected. Display fields are parsed just as they are currently with the SELECT command, that is, with all terms listed as single words, and classifications delimited by commas and semicolons.

In the following example, a term set is created containing each current classification (class/subclass) of all patents in answer set L1:

```
=> select L1 1-23 ccls
L2    SEL L1 1-23 CCLS:    83 TERM(S)
```

The system response indicates that 83 different classifications are listed on the 23 patents in answer set L1.

Searching a Term Set

Once created, an L-numbered term set can be displayed or searched; however, the term set can be searched only by itself, not combined with any other logic. Searching a term set searches the entire group of terms in the term set, connected by OR. For example, to search the file for all patents classified in one of these 83 current classifications, simply search the L-numbered term set (in this case, L2). This searches each class/subclass term in the term set, connected by OR logic:

```
=> s L2
L3    103   L2
```

Note that only the L-numbered term set is posted, not the individual terms. As the postings indicate, 103 patents in the file are classified in one or more of the 83 class/subclasses. You can now use the DISPLAY command to view those patents, or refine your search further by searching the new L-number (for example, SEARCH L3 AND ...).

Displaying aTerm Set

To view a list of terms in a term set, use the DISPLAY SELECT command (DISPLAY EXPAND or just DISPLAY may also be used). The resulting term set display includes a *term occurrence count* (how often each listed term occurs in the term set), a *document count* (the total number of documents in which the term occurs), and a *percentage count* (what percentage of documents out of the total term set contain the term).

Example term set display commands are shown below. Since the DISPLAY SELECT command is also used to list E-numbered terms resulting from a previous EXPAND or regular SELECT command, you must identify the term set L-number when specifying the DISPLAY SELECT command.

Command Example:	Result:
display select Ln entire	displays all of the terms in the term set
display select Ln top n	displays the first 'n' number of terms (e.g., TOP 10)
display select Ln n–n	displays the terms specified (e.g., 1-5)
display select Ln n–	displays an open-ended range of terms (e.g., 10–)
display select Ln ogt n	displays the terms with occurrence counts greater than 'n'
display select Ln dgt n	displays the terms with document counts greater than 'n'
display select Ln pgt n	displays the terms with percentage counts greater than 'n'
display select Ln %gt n	displays the terms with percentage counts greater than 'n'

The first time a term set is displayed, the default is *term occurrence order*, listing how often each term occurs, from most to least frequent. Term occurrence order can also be explicitly requested by specifying **'OCC'** in the command (for example, **DISPLAY SELECT Ln ENTIRE OCC**).

To display terms in *document count order* (listing the total number of documents in which each term occurs, ordered from highest to lowest), specify **'DOC'** in the command (for example, **DISPLAY SELECT Ln TOP 10 DOC**). To display terms in percentage count order (listing what percentage of documents in the term set contain each term, from highest to lowest), specify **'PER'** in the command (for example, **DISPLAY SELECT Ln TOP 10 PER**).

When an order is not specified, the terms are displayed in the same order as was used in the last display of the term set.

Sample DISPLAY SELECT results are are shown on the following pages, using a term set created by the following SEARCH and SELECT commands in the TRAIN file:

```
=> s (laser and beam)/ab
L4   24  (laser and beam)/ab

=> select L4 1-10 ab
L5   SEL L4 1-10 AB :   423 TERM(S)
```

The following shows a request to display all the terms in term set L5. Since this is the first time the term set is displayed, the default order is term occurrence order, from most to least frequent.

```
=> display select L5 entire
L5      SEL L4 1-10 AB :    423 TERM(S)

TERM  #OCC  #DOC  %DOC   AB
  1    40    10   100.00 BEAM
  2    31    10   100.00 LASER
  3    14     1    10.00 MEMBER
  4    11     1    10.00 SCREEN
  5    10     1    10.00 PUMP
  6     9     4    40.00 FIRST
  7     9     4    40.00 SECOND
  8     9     2    20.00 AXIS
  9     9     1    10.00 30
 10     8     4    40.00 BETWEEN
 11     8     1    10.00 REFERENCE
 12     7     6    60.00 A
 13     7     4    40.00 LIGHT
        •
        •
        •
```

The first line of this display indicates that BEAM, the most frequently occurring term in term set L5, appears a total of 40 times in a total of ten documents. These ten documents comprise 100 percent of the total number of documents (10) that were originally selected when term set L5 was created.

The entire term set list can also be displayed in order of highest to lowest document frequency count by specifying **'DOC'** in the command:

```
=> display select L5 entire doc
L5      SEL L4 1-10 AB :    423 TERM(S)

TERM  #OCC  #DOC  %DOC   AB
  1    40    10   100.00 BEAM
  2    31    10   100.00 LASER
  3     7     6    60.00 A
  4     9     4    40.00 FIRST
  5     9     4    40.00 SECOND
  6     8     4    40.00 BETWEEN
  7     7     4    40.00 LIGHT
  8     6     4    40.00 OUTPUT
  9     5     4    40.00 THROUGH
 10     7     3    30.00 SIGNAL
 11     6     3    30.00 USED
 12     5     3    30.00 SOURCE
 13     4     3    30.00 CAN
        •
        •
        •
```

Rather than display all terms, you can display just the first 'n' number of terms. The following example shows an Expert request to display the top 20 terms in term set L5. The default display order is document frequency count since it was the order last used:

```
=> d sel L5 top 20
L5      SEL L4 1-10 AB :    423 TERM(S)

TERM     #OCC #DOC %DOC  AB
   1      40   10  100.00 BEAM
   2      31   10  100.00 LASER
   3       7    6   60.00 A
   4       9    4   40.00 FIRST
   5       9    4   40.00 SECOND
   6       8    4   40.00 BETWEEN
   7       7    4   40.00 LIGHT
   8       6    4   40.00 OUTPUT
   9       5    4   40.00 THROUGH
  10       7    3   30.00 SIGNAL
  11       6    3   30.00 USED
  12       5    3   30.00 SOURCE
  13       4    3   30.00 CAN
  14       4    3   30.00 MEMORY
  15       3    3   30.00 DISCLOSED
  16       3    3   30.00 INCLUDES
  17       3    3   30.00 MADE
  18       3    3   30.00 PROCESS
  19       3    3   30.00 RESPONSE
  20       9    2   20.00 AXIS
38 MORE TERMS WITH A DOCUMENT COUNT OF 2
```

Note that since there are additional terms that are tied in the document count with the last term displayed, a message appears indicating how many additional terms have the same document count. If the number of terms with the same document count had been less than ten, the terms would have automatically been displayed.

The following example shows an Expert command request to display the top ten terms in term set L5. This time the display order is term occurrence count (most to least), explicitly requested by entering OCC. Since there is a tie in the occurrence count of the last (10th) term requested for display, the additional term that ties is also displayed (#11).

```
=> d sel L5 top 10 occ
L5      SEL L4 1-10 AB :    423 TERM(S)

TERM     #OCC #DOC %DOC  AB
   1      40   10  100.00 BEAM
   2      31   10  100.00 LASER
   3      14    1   10.00 MEMBER
   4      11    1   10.00 SCREEN
   5      10    1   10.00 PUMP
   6       9    4   40.00 FIRST
   7       9    4   40.00 SECOND
   8       9    2   20.00 AXIS
   9       9    1   10.00 30
  10       8    4   40.00 BETWEEN
  11       8    1   10.00 REFERENCE
```

The following example shows a request to display the top five terms in term set L5. By specifying PER, the display order is now based on percentage count, listing from highest to lowest the percentage of documents in the term set that contain each term:

```
=> d sel L5 top 5 per
L5        SEL L4 1-10 AB :    423 TERM(S)

TERM     #OCC #DOC %DOC AB
   1      40    10   100.00 BEAM
   2      31    10   100.00 LASER
   3       7     6    60.00 A
   4       9     4    40.00 FIRST
   5       9     4    40.00 SECOND
   6       8     4    40.00 BETWEEN
   7       7     4    40.00 LIGHT
   8       6     4    40.00 OUTPUT
   9       5     4    40.00 THROUGH
```

The top 9 terms are displayed since there is a tie in the percentage count.

You may also display terms with an occurrence count, a document count, or a percentage count that is greater than a specified number. The following example shows a request to display the terms in term set L5 with an occurrence count greater than 10:

```
=> d sel L5 ogt 10
L5        SEL L4 1-10 AB :    423 TERM(S)

TERM  #OCC  #DOC  %DOC AB
   1    40    10  100.00 BEAM
   2    31    10  100.00 LASER
  59    14     1   10.00 MEMBER
  60    11     1   10.00 SCREEN
```

Since no display order was explicitly requested, the previously used order of PER (percentage count) was used by default.

The following example shows a request to display, in highest to lowest document count (DOC) order, the terms in term set L5 with a document count greater than 3:

```
=> d sel L5 dgt 3 doc
L5        SEL L4 1-10 AB :    423 TERM(S)

TERM  #OCC  #DOC  %DOC AB
   1    40    10  100.00 BEAM
   2    31    10  100.00 LASER
   3     7     6   60.00 A
   4     9     4   40.00 FIRST
   5     9     4   40.00 SECOND
   6     8     4   40.00 BETWEEN
   7     7     4   40.00 LIGHT
   8     6     4   40.00 OUTPUT
   9     5     4   40.00 THROUGH
```

The following example shows a request to display the terms in term set L5 with a percentage count greater than 40. By default, the document count order is used:

```
=> d sel L5 pgt 40
L5      SEL L4 1-10 AB :     423 TERM(S)

TERM  #OCC  #DOC  %DOC   AB
1     40    10    100.00 BEAM
2     31    10    100.00 LASER
3     7     6     60.00  A
4     9     4     40.00  FIRST
```

For online information on listing terms in a term set, enter: **HELP DISPLAY SELECT** at an arrow prompt (=>).

To return to the default of creating an E-numbered list from the SELECT command, enter the following SET command:

```
=> set termset E#
```

For online information on the SET TERMSET option, enter: **HELP SET TERMSET** at an arrow prompt (=>).

Ability to List Classifications in an Answer Set

With the reloaded file, it is possible to obtain a list of all classification codes for an answer set. The SELECT command, which was not previously taught for use in APS text search, is used to obtain the list.

The command was not previously taught because the list of classifications obtained when using the command was not displayed in a useful format. Now, however, the software has been improved so that SELECT is useful.

Using the Select command, a list of classifications in an answer set can be obtained

Before Reload: There was no way to obtain a meaningful E-numbered list of classifications in an answer set.

After Reload: Using the SELECT command, an E-numbered list of classifications in an answer set can be requested. The examples shown on the following pages provide instruction for using this command.

(Continued)

Example

Search: In the Basic Index for the words HEAT PUMP and CONTROL. 19 answers are retrieved in answer set L14.

```
=> s heat pump and control
        1798  HEAT
         726  PUMP
          21  HEAT PUMP
                 (HEAT(W)PUMP)
        2971  CONTROL
L14       19  HEAT PUMP AND CONTROL
```

Select: The Novice version of the SELECT command is used. At the first two prompts, answer set L14 and answers 1 through 19 are requested.

The third prompt, DISPLAY FIELD CODE (TI):, is asking for a custom display format. The available formats are:

CCLS	-	Current Classification
ICLS	-	Issue Classification
FS	-	Field of Search
IPC	-	International Patent Classification

Current Classification (CCLS) is requested. The system responds with "E1-E61 ASSIGNED." This means that in all 19 patents in answer set L14 there were 61 different current classifications. Both original and cross-references are included.

```
=> select
ENTER ANSWER SET L# OR (L14): L14
ENTER ANSWER NUMBER OR RANGE (1): 1-19
ENTER DISPLAY FIELD CODE (TI): ccls
E1-E61 ASSIGNED
```

Note: The Expert version of the SELECT command in the example above would be entered as: **sel L14 1-19 ccls**

(Example continued on the next page.)

Example (continued)

Display Select: The Novice version of the DISPLAY SELECT command is used to request a display of all 61 classifications. E1 through E61 are displayed. The classification codes are ordered by number of occurrences (which is the same in this case as number of patents). Three of the patents out of the 19 selected are currently classified in 237/2B, three in 62/160, and so forth.

```
=> display select
ENTER (ALL), E#, RANGE, OR ?:  all
E1      3   237/2B/CCLS
E2      3   62/160/CCLS
E3      2   62/238.6/CCLS
E4      2   62/324.6/CCLS
E5      1   122/247/CCLS
E6      1   122/26/CCLS
 •
 •
```

Note:

1. The Expert version of the DISPLAY SELECT command shown in the example would be entered as: **d sel**

2. Rather than request a display of all classifications (by entering all at the prompt as shown in the example), you might want to request only the first 15 or so since they are displayed in order of occurrence. To request the first 15, at the prompt enter: **E1-E15**

The information obtained using DISPLAY SELECT can be used to provide direction for a paper search and/or for further text searching as shown on the following pages.

(Example continued on the next page.)

Example (continued)

Search: To isolate just the patents in L14 (out of the 19 retrieved) that are currently classified in 237/2B. The search is on a combination of E1, which represents class 237/2B, and the L-number for the answer set.

The intermediate posting shows that there are a total of four patents in the file that are classified in 237/2B. Since the L-number was included in the search statement, however, only three are retrieved.

Display: The citation information for all three patents.

```
=> s e1 and L14
          4   237/2B/CCLS
L15       3   237/2B/CCLS AND L14

=> d 1-3

1. 4,445,567, May 1, 1984, Thermostat for control of an add-on
heat pump system; Lorne W. Nelson, 165/29; 236/1E, 47, 78B;
237/2B [IMAGE AVAILABLE]

2. 4,385,725, May 31, 1983, Heat pump assembly; Franz
Pischlinger, 237/12.1; 122/26, 247; 237/2B

3. 4,232,820, Nov. 11, 1980, Heat collector system; Alfred E. Ritter, et
al., 237/2B; 62/235.1; 126/433; 237/1R
```

(Example continued on the next page.)

Example (continued)

Search: Of the entire file for E1 (representing class 237/2B). Since the L-number was omitted from the search statement, the entire file is searched and a total of four patents are retrieved.

Display: The citation information for all four patents.

```
=> s e1
L16        4   237/2B/CCLS

=> d 1-4

1. 4,445,567, May 1, 1984, Thermostat for control of an add-on heat pump
system; Lorne W. Nelson, 165/29; 236/1E, 47, 78B; 237/2B [IMAGE
AVAILABLE]

2. 4,385,725, May 31, 1983, Heat pump assembly; Franz Pischlinger,
237/12.1; 122/26, 247; 237/2B

3. 4,232,821, Nov. 11, 1980, Heating and ventilation system; Anders D.
Backlund, 237/2B; 62/325; 126/429, 431; 165/DIG.12

4. 4,232,820, Nov. 11, 1980, Heat collector system; Alfred E. Ritter, et
al., 237/2B; 62/235.1; 126/433; 237/1R
```

Delete: Before performing another SELECT, you may want to Delete your E-numbered list, as shown below using the Novice version of the command. This step is recommended because E-numbered lists obtained when using the SELECT command are not overwritten the way they are when using the EXPAND command. A subsequent Select will not start at E1 but will start after the last E-number on the previous list. The system limit for E-numbers in a list is 250.

```
=> delete select
DO YOU WANT TO DELETE ALL E# DEFINITIONS? (Y)/N:   y
ALL E# DEFINITIONS DELETED
```

Note: The Expert version of the DELETE SELECT command shown above is entered as: del sel

NETSCAPE

U.H.H.P

HAITIAN INTERNET DIRECTORY

The goal of this directory is to compile e-mail addresses and information from Haitians, Haitian Students, Haitian Scholars, Haitianists or anyone who has any interests or businesses in Haiti.

Ce repertoire electronique a pour but de compiler les adresses du courrier electronique Haitien et de creer un reseau de contact entre Haitiens, etudiants, chercheurs et quiconque interesse a Haiti.

Ralph Reid
Los Angeles, California USA
RafReid@Primenet.com
http://www.primenet.com/~rafreid

NEW REGISTRATIONS

Posted:8-7-95
Name: Dordy Joseph
e-Mail: DordyJ@aol.com
Location: Nyack, NY USA

546.99

Xterm

Posted:8-7-95
Name: Pierre Harboun
e-Mail: pd10u07@mailgtway.vpi.hydro.qc.ca
URL: http://www.vpi.hydro.qc.ca
Location: Montreal, Canada
Comments:I work as a computer professionnal for the Quebec Electric Utility Hydro-Quebec and from next month I will work for one year with the Haitian Electric Utility Electricite de Haiti - EDH

Posted:8-7-95
Name: Jephte Jeanniton
e-Mail: JJeannit@mit.edu
Location: Cambridge, MA
Comments: It gives me great pleasure to find about my country on the internet. I do hope that you will keep up the good work. PS. If anyone knows about a sofware that translate English to Creole and Vise Versa plese let me know I am really in need of one. I did hear that someone in New York had develope one, but I am not sure. Thanks.

Posted:8-7-95
Name: James K. McDonough

e-Mail: jmcdonou@oboe.calpoly.edu
URL: http://www.calpoly.edu/~jmcdonou/jim.html
Location: Shell Beach, CA, USA
Comments:I am looking for information on starting and conducting business in Haiti. Specifically I need to find out about obtaining a business license, what trade restrictions there might be, and any other legalities of doing business in Haiti. I am also looking for shipping information, import and export.

Posted:8-7-95
Name: Fritz Richmond **e-Mail:** aderbal@escape.com
URL: http://www.escape.com
Location: Brooklyn, NY
Comments:A Translator, speak fluent Haitian Creole. For all your translation don't hesitate to contact me.

Posted:8-7-95
Name: Jorn Sjostrom **e-Mail:** jorn.sjostrom@natmus.min.dk
Location: Copenhagen, Denmark
Comments: Father of two adopted boys, the one from Jamaica is 4 years old, 1995 the other from Haiti is 6 years old. I'm interested in knowing everything that has to do with Haiti and Jamaica to tell and show the boys. They are very interested in the countrys they come from.

Posted:8-7-95
Name: Steve C. Tozin
e-Mail: sct2@lehigh.edu
URL: http://www.lehigh.edu/~sct2/house.html
Location: Brooklyn, New York

Posted:8-2-95
Name: Urbain Louissaint
e-Mail: ulouissa@agt.net
Location: Calgary, Alberta, CANADA
Comments:A Computer Programmer-Analyst working with one of the leading Consulting Firms.

Posted:8-2-95
Name: Arold Norelus
e-Mail: anorel01@fiu.edu
URL: http://www.fiu.edu/~anorel01
Location: Miami, Florida
Comments:I'm a young haitian student currently majoring in chemistry at Florida International University. I'm interested in every issues that involves my country.

Posted:8-2-95
Name: Roberto J. Berrios
e-Mail: rberrios@netrunner.net
Location: Miami, Florida U.S.A.
Comments:I visited Haiti two months ago with a Fact-Finding Trade Mission organized by The City of Miami International Trade Board, the organization I work for. Since my specialties are trade research,

Posted:7-31-95
Name: Pierre E. Desir
e-Mail: okonkwo@escape.com
Location: Brooklyn,New York
Comments:24 years old, Grad student Majoring in Forensic Psychology and Law. Can Speak and write French, English and Creole. Can read and write Spanish.

Posted:7-31-95
Name: Steve Goodrich
e-Mail: sdgdrch@xmission.com
Location: Salt Lake City, Utah, USA
Comments:I lived in Haiti from Sept '84 to July '86. I miss Haiti and hope that I can go back some day with my family.

Posted:7-31-95
Name: Jean Laurent
e-Mail: jlaurent@soho.ios.com
Location: Hempstead, Nassau, New York
Comments: Together we can solve the economic problems of Haiti.

Posted:7-31-95
Name: Troy R. Samuels
e-Mail: trjls@aol.com
Location: Miami Florida
Comments: Served Mormon mission in Haiti from 1989 to 1991. Lived in Cap Haitien, Port au Prince and Petit Goave. Speak fluent Haitian Creole. Love Haitian food and music

---

Posted:7-31-95
Name: Jim Young
e-Mail: seaott@usit.net
Location: Nashville, TN
Comments: My parish in Nashville is a part of Haiti Parish twining Program. Our sister parish is St. John The Baptist in LaVallee. Would like to have any information about our sister parrish.

Posted:7-25-95
Name: Joseph Sosthene Geffrard
e-Mail: geffrard.J.S%wec@dialcom.tymnet.com
Location: Columbia, Maryland, USA
Comments: Bonsoi messie-dame la societe. Sak passe! I was born in Jacmel, but I am from many towns, and from ampil pays nan Haiti. I live to love Haiti and I love Haiti with passion. The two things I hate are tonton macoutes and zinglindos. I am a software engineer working for an engineering firm in Maryland. I am a computer science graduate of the university of Nebraska. Please drop me a note if you need my help, or advice or if you want to give me advice. Moin te l'ecol ka fre Jacmel, fre Gonaives, l'ecol Elliot Pierre, l'ecol George Marc. M'ap di bonsoi a tout zanmi'm, tout professor'm, tout moun ki te montre'm respecte tout moun. Viv democracy kap boujonnin en haiti. Foi sa'a se tout bon.

Posted:7-25-95

economic analysis, I was moved to study the trade links between Haiti & Miami which I believe should be strengthened due to its opening to Free Trade. It is my personal interest to contact local businessmen interested to develop commercial referrals with businesses in Miami, Florida as well as to develop trade among these countries in the auto parts industry. Please E-mail your interests for ways to better serve the trade community.

Posted:8-2-95
Name: Tammi L. McKinley
e-Mail: mckinlt@onrhq.onr.navy.mil
Location: Arlington, Virginia
Comments:I was a volunteer at the University of Les Cayes from May, 1991 through October, 1991. I would like to take my honeymoon in Haiti at the end of May, 1996, and need help with arrangements. I want to go back to Les Cayes, Jacmel, and Port-au-Prince.

Posted:8-2-95
Name: Troy R. Samuels
e-Mail: trjls@aol.com
Location: Miami Florida
Comments: Served Mormon mission in Haiti from 1989 to 1991. Lived in Cap Haitien, Port au Prince and Petit Goave. Speak fluent Haitian Creole. Love Haitian food and music.

Posted:7-31-95
Name: Harry P. Angus
e-Mail: 764721722@compuserve.com
Location: San Juan, Puerto Rico
Comments: Actually holding the position of Caribbean Operations Manager for leader in Transportation industry. I have plans to start importing and exporting to and from Haiti to the Caribbean and South America.

Posted:7-31-95
Name: Ben Berry
e-Mail: iftac@primenet.com
http://www.primenet.com/~iftac
Location: Northridge, California, USA
Comments: International Fund & Trade Assistance Company promotes international trade through providing funding and technical assistance to commisioned agents trading corporations and government agencies for all aspects of international trade.

Posted:7-31-95
Name: Paul Andre Cormier
e-Mail: pcorm00@mail.cpbx.net
Location: Columbus, Indiana, USA
Comments: Member of the US Coast Guard Port Security Unit. Had spent two and one half months in Port au Prince. Made many friends, mainly with the fishermen. Will be returning October 2, 1995 to visit my friends in Port au Prince. I am studying Haitian Creole. Would appreciate any assistance in answering the questions I have regarding the language and the people. Please e-mail me if you can help. Thanks, mesi anpil

---

Name: Ernst Louis Jacques
e-Mail: loui9244@mailserv.edmonds.ctc.edu
Location: Lynnwood, Washington
Comments:I am going back to Haiti on July 30, 1995. After this date, I can be reached by phone at (509) 46-5750, 46-1326, 23-6134

Posted:7-25-95
Name: Wesley Madhere
e-Mail: Wesley Madhere@turner.com
Location: Lithonia, Georgia

Posted:7-25-95
Name: James Cofield
e-Mail: james.cofield@internetmci.com
Comments:I am currently developing marketing programs for tele-medicine programs, electronic classroom training, video conferencing (room & desktop technologies),commodities trade and computer technologies. Looking for partners for Haitian partners to develop markets. Voice 814 472-3146 Fax 814 472-3377. Home 202 829-1718

Posted:7-24-95
Name: Hermann Heraux
e-Mail: HermyH@aol.com
Location: Chicago, USA
Comments: Let us unite for a wired Haiti. We missed the industrial revolution, let's embrace the Information revolution

---

Posted:7-24-95
Name: Lionel Martelly
e-Mail: lmartelly@aol.com
Location: Mill Neck, USA
Comments: Will talk to any executive in the Haitian garment industry.

Posted:7-24-95
Name: Mary Lorimer
e-Mail: mlorimer@ix.netcom.com
Location: Montcliar, NJ. USA
Comments: On behalf of a friend not yet on the net, Jean-Edward Jeremie, I am searching for Haitian Professional Organizations%2FAssociations in the New York Metro area. Thank you

---

Posted:7-24-95
Name: Robert J. S.
e-Mail: alexzoe@aol.com
Location:NYC, NY
Comments: Sa ki pa kanpe djanm, fe wout ou

Posted:7-22-95
Name: Garry Jean-Louis

e-Mail: gjeanlou@acs.ucalgary.ca
Location: Calgary, Canada
Comments: Haitian citizen, strong supporter of the cause of justice

---

Posted:7-22-95
Name: Nicola Virgill
e-Mail: nivirgill@vassar.edu
Location: Poughkeepsie, NY USA
Comments:I am a Bahamian studying Economics at Vassar College. I have a great interest in learnign more about Haiti and Haiti's economy. Feel free to write me on any subject.

Posted:7-22-95
Name: Frederica Stines
e-Mail: stines@unv.ch
Location: Geneva, Switzerland
Comments:I am a Haitian American interested in Haitian issues and disscussions.

Posted:7-22-95
Name: Trevor W. Purcell
e-Mail: purcell@chuma.cas.usf.edu
Location: Tampa, Florida, USA
Comments:I am Assoc. Prof. of Anthropology and Africana Studies at the University of South Florida. Areas of specialty: Caribbean; palnned social change; political economy; culture; ideology

---

Posted:7-22-95
Name: Lionel Martelly
e-Mail: LionelM650@aol.com
Location: New York, USA

Posted:7-22-95
Name: Christer Joensson
e-Mail: christer.jonsson@buller.se
Location: Laholm, Sweden
Comments:45 years of age. Product Manager at a software company in Sweden. Develops software for school administration. Market leader in Sweden. Also established in Germany, Norway, Denmark, Belgium and Russia.

Posted:7-22-95
e-Mail: croehl@wctc.net
Comments: Our interest in Haiti resulted from our linking with St. Joseph Parish in Bonneau, Haiti through the Haiti Parish Twinning Program. We'd appreciate any information anyone has about the Bonneau area.

Posted:7-22-95
Name: Stephane d'Amours
e-Mail: interso@cam.org
URL: http://www.cam.org/~interso/index.html

Location: Montreal, Canada
Comments: Marketing and communication for human development. In my virtual office WEB site you will find all the information related to me some links related to international development, private enterprises and organizations involved in international development... Also, you can register to my web by sending an e-mail.

Posted:7-20-95
Name: Erick Bourraine
e-Mail: Boubou@ids.net
Location: Miami, Florida, USA
Comments:I am involved with a marketing company catering to any organization business or individual aspiring to reach the Haitian population in the US.

Posted:7-20-95
Name: Pierre Abel
e-Mail: cecqint@edupac.qc.ca
Location: Quebec, Canada

Posted:7-19-95
Name: Florida A&M University Haitian Cultural Club
e-Mail: famuhcc@freenet.tlh.fl.us
Location: Tallahassee, Florida U.S.A.
Comments: The Florida A&M Haitian Cultural Club is located in Tallahassee, Florida U.S.A. and coordinates it's activities with students at Florida State University and Tallahssee Community College. Combined, we have over 100 members and are looking for any ideas for Haitian Cultural week. One of our goals is to promote our culture in America and her people. If there are other student organizations out there, we would love to hear from you. President: Chantale Fortunat, Treasurer: Roberto Thomas Phone: (904) 599-3736 or (904) 681-0137

Posted:7-19-95
Name: Valerie Mamara
e-Mail: vamm@strauss.udel.edu
Location: Newark, Delaware, USA
Comments:I am a graduate student studying agricultural economics. I am very interested in rural economic development in Haiti. I have a special place in my heart for Haiti and its people because in 1992 I had the opportunity to work with Haitian youth just arrived from Guantanamo in Jackson, MS.

Posted:7-17-95
Name: Gerald L. Boarino
e-Mail: glb@daka.com
Location: Port Townsend, WA USA
Comments: Retired university professor (romance langs/literatures) who is conversant with haiti. Am interested in promoting haiti's philately (stamps, postal history, etc). If interested in info., please contact. Plan on a trip to haiti this late fall. Anyone in haiti at moment who has contact with UN troops (not US)? Need to contact for philatelic purporses (article that I am working on).

Posted:7-17-95

Name: Ronald Frische
e-Mail: RFrische@aol.com
Location: Kansas City MO U.S.A.
Comments:I am interested in importing Haitian seafood to the USA.

Posted:7-17-95
Name: Patrick Bellefleur
e-Mail: pbellefl@netrunner.net
Location: Miami, Florida, USA
Comments: Now, Haitian all over the world can get together over the Internet.

Posted:7-15-95
Name: Franck Daphnis, Jr.
e-Mail: emcleod@worldbank.org
Location: Washington DC, USA
Comments:I currently work for the Cooperative Housing Foundation (CHF) where I manage an urban sanitation project in Port-au-Prince, Haiti (technical assistance and jobs creation). I would be interested to hear from any compadres (or otherwise interested Haitianophile) on ideas, comments and gripes regarding urban issues in Haiti. I went to high school in Haiti (St Louis) and studied architecture (Wisconsin) and city and regional planning (Cornell) in the US. Drop me a note (this is my wife-to-be's e-mail address).

Posted:7-15-95
Name: Amy Waldorf
e-Mail: awaldorf@recsports.calpoly.edu
Location: San Luis Obispo, California, USA
Comments:I am working on a project to help improve the Haitian economy. I am interested in any information regarding business practices, product or service needs in Haiti, and suggestions about how to employ the citizens. I am working with a company who is willing to invest money in a business to help stimulate the Haitian economy if we can form a business plan that will be successful.

Posted:7-15-95
Name: Weedens E Blanchard
e-Mail: AKPJSJU@stjohns.edu
Location: New York, USA
Comments:I am a Haitian Graduate Student at St. John's University in New York City. I received my Bachelor of Science degree in Computer Science and I am now pursuing my M.B.A degree in Executive Management. If there is anyone wanting to get on-line to speak about sports, education, politics, computers, or any other subject, please email me. I am always interested in what other individuals have to say concerning these topics.

Posted:7-15-95
Name: Ron Curl
e-Mail: 76717.562@compuserve.com
Location: Cambridge, Ontario

Posted:7-15-95

Name: Deborah Curl Bowers
e-Mail: dbowers@ucs.indiana.edu
Location: Bloomington, Indiana
Comments: Grew up in Haiti, attended Quesqueya Christian School.

Posted:7-15-95
Name: Fritz S. Dubuisson
e-Mail: Dubuisson@wit.edu
URL:http://www.wit.edu
Location: Boston, MA USA
Comments: Civil Engineering student at W.I.T, also pursuring a career in music specially Jazz. Heavily influence by Charlie Parker.

Posted:7-13-95
Name: Hansy Martelly
e-Mail: HMartelly@aol.com
Location: Pasadena, California,USA

Posted:7-13-95
Name: Laura Dooley
e-Mail: dooleyl@uthscsa.edu

Posted:7-10-95
Name: Romuald Jadotte
e-Mail: Tachou@aol.com
Location: Tampa, Florida, USA
Comments:I am a Haitian-American entrepeneur. I own a small computer business specializing in networking, Lan installation, system upgrade and computer repair. Eventually I would like to expand my business to benefit the Haitian community here and back in Haiti. It's great to finally have a Haitian forum on the Web.

Posted:7-10-95
Name: Harry Evans
e-Mail: hevans@ideal.ios.net
Location: San Diego, California, USA
URL: http://ideal.ios.net/~hevans/home0.html

Posted:7-10-95
Name: Gui Pradieu
e-Mail: Guifaurep@aol.com
Location: Brooklyn, NY
Comments:I am an Industrial Engineer working for a multinational foods company. My professional interests are industrial engineering, manufacturing, TQM.

Posted:7-10-95
Name: Nancy Robinson
e-Mail: nancyrob@aol.com

Location: Stratford, Connecticut, USA
Comments: First generation Haitian-American.

Posted:7-8-95
Name: Jean-Robin Jadotte
e-Mail: JJ76@columbia.edu
Location: New York, N.Y. U.S.A.
Comments:I'm looking for a friend: Patrick Scott last I heard from him was in jan 95. He was heading to Haiti with his wife and son. If anybody see or know him ask him to call 212 854-4363. Merci

---

Posted:7-7-95
Name: Reginald Charlemagne
e-Mail: charlemr@jsp.umontreal.ca
Location: Montreal, CANADA.
Comments: Je suis etudiant dans un Bacc Specialise en Informatique a L'Universite de Montreal. Je suis particulierement interesse a tout ce qui a rapport avec cette branche. Si vous avez des informations interessantes a partager a ce sujet, je suis deja sur le point de repondre a votre courrier. Mais je suis tout aussi ouvert a tout genre de sujets.

Posted:7-7-95
Name: Vladimir J. David
e-Mail: vdavid@cvbnet.cv.com
Location: Bedford, Massachusetts
Comments: Je suis Ingenieur en Electronique et je travail a present comme Ingenieur de Langue en Ordinateur pour Computervision. Quiconque peut utiliser ma connaissance au profit de leur organisation en Haiti peut toujours me contacter toute les fois que l'offre est d'une certaine durabilite.

Posted:7-7-95
Name:WALKER SAINVIL
e-Mail: walkers329@aol.com
Location: Miami, Florida USA
Comments:I am planning to start a film festival in Haiti...If anyone knows where I can get resources and sponsors please contact me.

Posted:7-4-95
Name: Jean-Marie Bourjolly
e-Mail: jean-m@crt.umontreal.ca
Location: Montreal, Quebec, CANADA
Comments:I am an Associate Professor of Operations Research Production Management at Concordia University. My professional interests include the study of classical models and methods of O.R. and their applications to telecommunications and production planning as well as teaching and curriculum design.

Posted:7-3-95
Name: Will Herman
e-Mail: willherm@ix.netcom.com
Location: Los Angeles, CA, USA

Comments:I am an American who has been to Haiti several times with many pleasant memories. I am interested in keeping in touch with Haitians and friends of Haitians. Drop me a line.

Posted:7-3-95
Name: Anita Snow
e-Mail: anita@mail.internet.com.mx
Location: Mexico City, Mexico
Comments:I am a journalist based in Mexico City and travel to Haiti two or three times a year to help our reporters covering the caribbean. My area of expertise is Mexico and Central America, but I've grown fascinated with Haiti. I would like to keep up with what is happening in Haiti during a year I believe will prove to be crucial to the nation's uture.

Posted:7-3-95
Name: Gilles Lacasse
e-Mail: edith@prof.synapse.net
Location: Quebec, Canada
Comments: Je suis un enseignant passionne de telematique. J'enseigne depuis 30 ans dans les ecoles secondaires au Quebec, Canada. J'ai travaille comme benevole a Industrie Canada. Cette annee j'etais en pret de services avec le Reseau scolaire canadien (Schoolnet). Je serais interesse a me rendre en Haiti. Pour la prochaine annee je serai en annee sabbatique. Connaissez-vous des gens qui pourraient m'aider ?

Posted:7-3-95
Name: Rev. Dan Jenkins
e-Mail: wtknight@pgh.nauticom.net
Comments: Bonjou tout zanmi mwe. Map chache pou let ou! Mwe vle konne nou tout. M'gin place Good News Cafe. Vini nou, non? Visite-m. Esk ou konne Pasteur Preval Floreal? Pasteur Megy Sylvain? Pasteur Thimote Sylvain? Bon zanmi-m. Viv Ayiti! Beni swa L'Eternel! Nan non Roua Jezikri, Pasteur Daniel.

Posted:6-30-95
Name: Serge D'clama
e-Mail: 74227.635@Compuserve.Com
Location: Decatur, Georgia, USA
Comments:I Publish The Creole Connection a newsletter on Business Development or opportunities in Haiti. I would love to receive information that could be of interest to my readers.

Posted:6-30-95
Name: Kenneth Bever
e-Mail: KBever_+a_ho_+lKenneth_Bever+r%HSB@mcimail.coml
Comments:I am the director of "Hope for Haiti's Children" -- a new, direct Haitian child sponsorship program. The purpose of Hope for Haiti's Children is to meet the basic educational, health, and spiritual needs of very poor children in Haiti through a direct sponsorship program run by volunteers. A primary goal of this non-profit program is to keep overhead costs to an absolute minimum. None of the $22 a sponsor contributes per month is used for promotional materials, fund-raising, or salaries. All the money goes into 6 different areas to help your child: tuition, school supplies, school clothing, food, medical care (preventive & emergency), and communication (letters to and from your child). If you are interested in more information, just send me an e-mail and I'll send you more information.

Posted:6-30-95
Name: Dominique J. M. Northecide
e-Mail: dom@dev.cdx.mot.com
Location: North Attleboro, MA USA

Posted:6-30-95
Name: Firmin Backer
e-Mail: backer@saddle.anlw.anl.gov
Location: Idaho Falls, Idaho, USA
Comments:I am an engineer working for the U.S. Department of Energy at Idaho Engineering Laboratory. I am from New York where I have been for the pass 14 years. I will be at this address for the next 6 weeks and then. I wll be returning to New York. I am happy to be part of this Haitian internet directory. Please anyone drop me a note. Your fellow Firmin Backer

---

Posted:6-28-95
Name: Felix Saint-Victor
e-Mail: T91_SAX@t.kth.se
Location: Stockholm, Sweden
Comments:I'm halv Haitian. My other halv is Norwegian and I was born in Boston (1969). My mother is from a little town in the west of Norway, and my father is from Port-Au-Prince. In 1972 we moved to Haiti. We lived at Fontamara 27. I went to school at Au Galop and College Canado. In 1987, I moved to Norway. I took high school over again in order to be able to learn the language and get into the system here in scandinavia. In 1991, I moved to Sweden to study Aeronautical Engineering. After the two first years, I decided to change field. I went into Industrial/Systems Engineering. I'll finish my education around January 96. I really don't have any specific plans for the futur. I'll probably start working in Norway. I do have Haiti in mind (and in my heart), but right now I think things are to hot!

Posted: 6-23-95
Name: Ancy Verdier
e-Mail: averdier@emerald.tufts.edu
Location: Somerville MA
Comments:I am currently the president of the Tufts student body, I'm in my last year at Tufts and I'm planning to go to med school. If anyone can give me any info on med school that would be appreciated. My address is 71 Raymond Ave. Somerville MA, 02144

Name: Dr. Louis Larosiliere
e-Mail: splouis@poop.lerc.nasa.gov
Location: Cleveland, Ohio
Comments: I am a Haitian-American. It is a great priviledge to learn about my heritage via this "high-tech" information exchange. I am a mechanical engineer (BS/MS/PhD) engaging in fluid dynamics research at NASA Lewis Research Center. My goal is to contribute to genuine development of Haiti and the Haitian people. phone: 216-433-3403 NASA Lewis Research Center, Cleveland, Ohio 44135

PAST REGISTRATIONS

Secondly, it could potentially be used to very easily search your total patent database for prior art at the patent level or at the claim level. The distance measure we used to determine the nearness of other patents can also be used at the individual claim level. It could be available to you to automatically create responses to patent applications.

The third potential use would be in your non-patent literature. It would only be a case of setting thresholds to eliminate consideration of NPL, since we know our method forces the relationship of any non-patent literature into one of the categories.

Although the fourth area does not appear valuable to us, it may to you. The consistency between the specification and the claims could be checked on an automatic basis.

These are simply some random ideas as to how the existing technology could be applied in support of your activities. It would probably create significant increases in productivity. This often creates problems in American industry today, and I am sure, in the government sector as well. However, with all of these considerations, we hope you find this useful. Our invitation to visit our lab in Albuquerque is always available to you. If there is some cooperative effort that you feel we should do together, feel free to give us a call.

Sincerely,

Glen Self

Glen Self
Vice President, R&D

cc: Tom Giammo
Bill L. Lawson

Fax Transmission

No. of pages incl. this one: 5

To: **Ed Earls**

Fax number: Voice:

cc:

From: William S. Lawson, Administrator

Date: Wednesday, February 3, 1993

Bill

If you do not receive all pages, please contact:

Search & Information Resources
Administration; CPK3-702
U.S. Patent and Trademark Office
Washington, D.C. 20231
703-557-0400/703-557-0668 (FAX)

Subject:

Message: *Ed:*

Attached is information about the EDS software. Please have Duane look it over, in particular to see what we need that we don't have, e.g, Sybase for the SUN.

I'd like to get together next week to discuss this matter in general. Would be helpful if, at that time, Duane is prepared to react to this document.

if

Willette will set up the meeting

USPTO Patent SW Configuration

This paper describes the configuration requirements for the Patent Tool software designed by the EDS Research Albuquerque Lab and used in the USPTO tests. The Patent Tool is a collection of programs and scripts designed to run various tests, including Frequency Bayes Categorization.

Here are some highlights.

Hardware:

• Sybase	4 GB of disk space per year of patents
	1 GB of disk space for the 7210 patents
• FreqBayes Results	Highly variable: 5 MB to 100 MB per test (or more)
• ClusterTool	1095 patents -> 19 MB disk space
	1095 patents -> 44 MB memory
• What's New programs	10 machines for USPTO missed test. What's New (and Mortar) hardware configuration vary depending on use.

Software ≙ PATENT SW ?, PATENT TOOL

• Sun OS 4.1.x	
• GNU g++ & gcc	Provided on tape
• csh, awk, & perl	provided by system or by tape
• X11	
• InterViews 2.5 & 2.6	Provided on tape — ivedit for PViewer
• Sybase	
• In-House libraries	Provided on tape
• ClusterTool	Currently only an executable is provide on tape. — GCanvas Editor
• What's New (Mortar)	
• { SupLibs?	⟹ What about the "In-House" libraries needed to build the executable file.

1.0 Hardware

Hardware for the Patent Tool consists of machines and disks to store the patents and any results generated. It also consists of machines to run the individual programs and scripts.

1.1 Database Requirements

The patents used in the tests need to be stored in a database of some form. Currently, Sybase is used to hold the patents. Not all information concerning a patent is kept in the database. Several small fields, such as patent number,

date issued, assignee, etc., are kept along with the title, abstract and claims. For patents in 364 the summary and description (examiner background) is kept. The 1989 patents consume 1.5GB of disk space. If the summary and description information was kept for all patents, 1989 patents would consume an estimated 4 GB of disk space. The above data does not include any results created by the Patent Tool.

The size of the results from the Frequency Bayes test is highly variable. A test with the 7210 patents consumed near 100 MB. After removing everything except the actual categorization results, 5 MB of disk space was consumed. The files removed consisted of temporary files, files used by the PViewer to view the patents, and the histogram files for each patent.

The Cluster Tool creates a distance matrix. This matrix is stored on disk. For the Bayes3 clustering (691 patents), the file was 8MB. For the USPTO clustering (1095) patents, the file was 19MB. If the patent text is to be viewed, it must reside in a directory the Cluster Tool can find. This increases the disk space needed to run the ClusterTool. The actual amount depends on how many patents are used and what portions of the patents are to be viewed.

1.2 Program Requirements

The programs extract data from the Sybase database, or a unix directory or file and then process the data. Depending upon the number of patents the amount of memory will vary. The Cluster Tool used 44 MB of memory when clustering the USPTO category 12 (1095 patents). If the machine the process is running on does not have enough real memory, the program will thrash causing a tremendous slow down. For large tests, lots of real memory is needed.

The What's New programs actually need several machines to run effectively. In the test to find patents from 1989 and 1990 that are candidates for class 395 possibly missed by the USPTO, ten categorizers were run on ten different machines. This test uses Mortar. Mortar itself ran on yet another machine. Ideally, no categorizers should run on the machine running Mortar or the machine running the Sybase server.

The What's New release tape includes Mortar. Many varied hardware configurations can be created. The use determines the hardware configuration. It is possible to use only one machine, but it is recommended that the minimum configuration should include at least three machines.

2.0 Software

The programs were compiled on a Sun4 running Sun OS 4.1.1 using the GNU g++ and GNU gcc compiler. In addition to the actual programs, several support routines from various libraries written by the EDS Research Lab are used. An attempt to remove all dependencies outside of the support library, the GNU g++/gcc compiler, and Sun OS 4.1.1, and those stated below has been made. The attempt may or may not have been successful.

Most scripts use the csh shell, others awk, and still others perl.

Following is a breakdown of configuration dependencies not describe above for individual portions of the Patent Tool.

2.1 Frequency Bayes Categorization

The Frequency Bayes Categorization uses Slurp to retrieve patents from a Sybase database.

Currently, data is maintained in a format compatible with the Personalized Viewer (PViewer). The PViewer (if used to view patents) is dependent on the ivedit library. This is a support library written at the EDS Research Albuquerque Lab. The ivedit library requires X11 and InterViews 2.5.

Software Dependencies	Description of Software
Slurp	Database Interface
Sybase Library	
tools library	general purpose routines
PViewer	Browsing Patents
ivedit Library	Editor library
whn_lib	Unix What's New library (Clipping Service)
InterViews 2.5	
X11	
tools library	general purpose routines

All of the above software dependencies, except for X11, is included on the PatentSW release tape.

2.2 Cluster Tool

The Cluster Tool requires several of the support libraries from the Albuquerque Lab. Included in the required libraries is GCanvas, and Editor. These

[which] [and]
libraries depend on X11 and InterViews 2.6. These two libraries may have other dependencies.

Software Dependencies	Description of Software
GCanvas InterViews 2.6 X11	Graphical software package
Editor InterViews 2.6 X11	Editor software package
tools library	general purpose routines
whn library	Unix What's New library (Clipping Service)

The Cluster Tool program currently is distributed as an executable. The actual source for the program is included, but not all of the in-house libraries are included in the Support Library release (SupLibs) so the executable cannot be built.

2.3 What's New Test

The What's New test, finding additional candidates for class 395 that the USPTO possibly missed, requires the What's New specialists and the Mortar System. The What's New release tape contains a binary and a source code release of What's New, Mortar, and the Support Libraries. This is not included with the PatentSW release tape.

Summary of Personnel Qualifications.

Joe R. Hill

Education:

Ph.D., Mathematics, 1986, University of Texas, Austin, TX, Dissertation: Empirical Bayes Statistics: A Comprehensive Theory for Data Analysis.

MA, Mathematics, 1982, University of Texas, Austin, TX, Thesis: Improving Failure Rate Estimation Using Parametric Empirical Bayes.

BS, Mathematics, 1980, University of Texas, Austin, TX

Experience:

1986 to present: EDS Research, Albuquerque, NM, R&D Specialist

1980-1986: University of Texas, Austin, TX, Research Assistant

Pete Humphrey

Education:

BA, Linguistics, 1976, University of New Mexico

Experience:

1985 to present: EDS Research, Albuquerque, NM, R&D Specialist

1980-1985: Excalibur Technologies Corp., Albuquerque, NM, Software Engineer

1977-1980: University of New Mexico Computing Center, Albuquerque, NM, Systems Programmer

Melissa A. Macpherson

Education:

MA, Linguistics, 1988, University of New Mexico

Thesis: Individual Syntactic Variation: the Relative Clause in English.

BA, Linguistics, 1983, University of New Mexico

Experience:

1988 to present: EDS Research, Albuquerque, NM, R&D Specialist

1983- 1985: University of New Mexico, Teaching Assistant

Sue A. Medeiros

Education:

MS, Computer Science, 1981, University of New Mexico, Albuquerque, NM

BA, German, 1970, University of Hawaii, Honolulu, HI

Experience:

1986 to present: EDS Research

1984-1986: Sperry Corp., Albuquerque, NM, Systems Software Engineer

1983-1984: University of New Mexico, Albuquerque, NM, Lecturer

1981-1983: Boeing Military Aircraft Co., Seattle, WA, Software Engineer

1980-1981: University of New Mexico, Albuquerque, NM, Research Assistant

1979-1980: University of New Mexico, Albuquerque, NM, Teaching Assistant

Mark Schnedar

Education:

BS, Computer Science, 1986, University of New Mexico, Albuquerque, NM

Experience:

1985 to present: EDS Research, Albuquerque, NM, R&D Specialist

1985: IBM Corp., Austin, TX, Student Intern

H. Kelly Shuldberg

Education:

Ph.D., Linguistics, 1987, University of Texas, Austin, TX.
Dissertation: Syntactic and Semantic Issues in Second Language Learning.

MA, English, 1981, Utah State University, Logan, UT.
Thesis: An Examination of Focus on Form for Monitor Function.

BA, English, 1978, Utah State University.

Experience:

1990 to present: EDS Research, Albuquerque, NM, R&D Specialist

1989: Eastman Kodak Co., Rochester, NY, Computational Linguist

1987-1989: Automated Language Processing Systems, Salt Lake City, UT, Computational Linguist

1985-1987: Microelectronics and Computer Technologies Corp., Austin, TX, Research Intern

William C. Slade III

Education:

BS, Computer Science, 1973, New Mexico State University, Las Cruces, NM

Experience:

1985 to present: EDS Research, Albuquerque, NM, Division Manager

1980 -1985: Excalibur Technologies Corp., Albuquerque, NM, Software Development Manager

1976-1980: University of New Mexico Computing Center, Albuquerque, NM, Systems Programmer

1974-1976: Harris Data Communications, Dallas, TX, Systems Engineer

Anne M. Tomasi

Education:

BS, Computer Science, 1987, University of New Mexico, Albuquerque, NM

Experience:

1988 to present: EDS Research, Albuquerque, NM, R&D Specialist

Greg L. Whittemore

Education:

Ph.D., Linguistics, 1987, University of Texas, Austin, TX

BS, Speech Communications, 1978, University of Texas, Austin, TX

Experience:

1989 to present: EDS Research, Albuquerque, NM, R&D Specialist

1987-1989: Eastman Kodak Co., Rochester, NY, Research Scientist

1985-1987: Microelectronics and Computer Technologies Corp., Austin, TX, Research Intern

Ed

In my opinion, these tests are inconclusive primarily for two reasons:

1. EDS chose to have their system "learn" from our complete set of classified patents rather than from a sample. (Self says in his letter, "we felt that a training set of approximately twenty per category would have been sufficient." So why didn't they work with just twenty per category? This is exactly what we had hoped to see demonstrated.)

2. Apparently, EDS ignored our rule of hierarchy, and therefore its not clear what the comparison of their results with ours really indicates.

The EDS results of their first run (Bayes0) is the most realistic for purposes of comparison with the PTO classification data. This is because the EDS precision and recall results for the subsequent runs--Bayes1,2 and 3--are based on Bayes0,1 and 2, respectively, rather than on the PTO classification data. Thus, because they ignored our hierarchy rule, Bayes1,2 and 3 would inevitably diverge more from the PTO data than Bayes0. Therefore, the best EDS results would appear to be 77.3% for overall accuracy with individual subclass recall and precision percentages as low as 59% and 64%, respectively. (See page 5.) Not very impressive in view of the fact that the EDS system used the entire PTO database to "learn" from.

As for "the problems with overlapping categories" referred to on page 5, this seems to be the result of EDS ignoring our hierarchy practice. In other words, this may be a problem for EDS, but not a problem for the USPCS.

Regarding the "additional patents that the USPTO missed," (see pages 19 and 22+), note that most of these are from subclasses in the project. Thus, I suspect that these patents were among those not yet keypunched when the PTO database was created.

On the other hand, the results for the "Bayes0" test on the two-dot indents under "Presentation Processing" are impressive: 96% overall with a minimum of 90% for both recall and precision! (See pages 43+.) The real question--from the classifier's point of view--however, remains unanswered: how accurate is the system when it "learns" from less than the whole reclassified database? This may be a question we could answer ourselves.

K Orr

9-26-91

cc. Bill Lawson

USPTO Class as Training Set – Adjusted for hierarchy of USPTO.

Bayes 0

USPTO			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Recall	
			19	3	4	6	16	10	11	12	13	14	15	8	9	7	5	1	2	17	18		
0-0	19	(435)	308	8	4	16	1	2	2	4	12	2	1	4	8	29	15	2	5	1	11	71	NC
1-1	3	(230)	2	189		15	1		2		1	6			3	5	5		1			~~82~~	100
2-2	4	(245)	7	2	179	12	1		5		2	2		2	6	11	10		4		2	73	98
3-3	6	(639)	31	23	12	461	14		9	4	2	9		2	23	7	36			1	5	72	89
4-4	16	(211)	2	3		4	171	2	1	2		4	1	1	4	9	3		1	2	1	81	83
5-5	10	(150)			2	4	1	119	2		2			1	4	4	8	1	2			79	83
6-6	11	(235)	3	4	3	6			201			3			4	6			1	1	3	86	90
7-7	12	(343)	4	1	7	10	3	3	9	242	6	8	1	2	14	8	10	2	2	4	7	71	80
8-8	13	(177)	1	1		1					164	2		1	1		2		4			93	94
9-9	14	(184)	2	1		1	3		2		1	162	3	1	1	5			1		1	88	93
10-10	15	(87)								3	1	1	72		2		4		1	3		83	86
11-11	8	(338)	7		2	3			1	1	1			273	20	19	1		1		9	81	92
12-12	9	(1095)	29	7	31	57	22	6	26	20	25	9	3	82	649	41	39	3	19	6	21	59	75
13-13	7	(607)	12		3	3	3	1	3	2	2	8	5	9	4	524	6		2	1	19	86	98
14-14	5	(551)	7	7	12	28	27	3	4	8	4	5	2	4	11	7	393	1	13	2	13	71	94
15-15	1	(442)	5	2				2	2		4	1			4		1	415	6			94	100
16-16	2	(946)	6	1		1	1	4	5	8	19	3	7	1	11	2	31	7	834		5	88	99
17-17	17	(47)																		47		100	NC
18-18	18	(248)	2	3	2	5	3		6	5	1	4	1	3	9	29	3		2	1	169	68	69
Totals:		7210	428	252	257	627	251	142	280	299	247	229	96	386	778	706	567	431	899	69	266	5572	
Precision:			72	75	70	74	68	84	72	81	66	71	75	71	83	74	69	96	93	68	64		
Total horizontal			308	229	239	568	176	124	211	275	167	171	75	292	816	593	490	442	939	47	170		
additional vert.			+120	+ 3	+ 0	+56	+76	11	49	+43	68	56	21	22	82	32	47	0	6	21	86		
total/cat.			428	232	239	624	252	135	260	318	235	227	96	314	898	625	537	442	945	68	256		
			72%	99%	100%	91%	70	92	81	86	71	75	78	93	91	95	91	100	99	69	66		

% of pats in Bayes categories which correspond to patents in PTO sub, including pats rolled up from inferior cats.

% of pats in all Bayes cats, less pats in superior cats, relative to pats in corresponding US sub.

our SUBS in hier. order | Bayes0 Recall } % of patents placed in our subs by EDS relative to total number of pats. we placed in sub.

Bayes 0 Precision } % of pats placed in our subs that were the same as the pats we placed in our subs.

our SUBS in hier. order	Bayes0 Recall	Bayes 0 Precision
1	100	100
2	99	99
3	100	93
4	98	100
5	94	89
6	89	89
7	98	94
8	92	94
9	75	89
10	83	88
11	90	77
12	80	85
13	94	71
14	93	74
15	86	77
16	83	69
17	100	70
18	69	66
19	71	72

Retabulated taking into account our hierarchy

USPTO Class as Training Set

Bayes 0

USPTO	our hierarchical order		0 (19)	1 (3)	2 (4)	3 (6)	4 (16)	5 (10)	6 (11)	7 (12)	8 (13)	9 (14)	10 (15)	11 (8)	12 (9)	13 (7)	14 (5)	15 (1)	16 (2)	17 (17)	18 (18)	Recall	
0-0	19	(435)	308	8	4	16	1	2	2	4	12	2	1	4	8	29	15	2	5	1	11	71	NC
1-1	3	(230)	2	189		15	1		2		1	6			3	5	5		1			82	100
2-2	4	(245)	7	2	179	12	1		5		2	2		2	6	11	10		4		2	73	98
3-3	6	(639)	31	23	12	461	14		9	4	2	9		2	23	7	36			1	5	72	89
4-4	16	(211)	2	3		4	171	2	1	2	4	4	1	1	4	9	3		1	2	1	81	83
5-5	10	(150)			2	4	1	119	2		2			1	4	4	8	1	2			79	83
6-6	11	(235)	3	4	3	6			201			3			4	6			1	1	3	86	90
7-7	12	(343)	4	1	7	10	3	3	9	242	6	8	1	2	14	8	10	2	2	4	7	71	80
8-8	13	(177)	1	1		1					164	2		1	1		2		4			83	94
9-9	14	(184)	2	1		1	3		2		1	162	3	1	1	5			1		1	88	93
10-10	15	(87)								3	1	1	72		2		4		1	3		83	86
11-11	8	(338)	7		2	3			1	1	1			273	20	19	1		1		9	81	92
12-12	9	(1095)	29	7	31	57	22	6	26	20	25	9	3	82	649	41	39	3	19	6	21	59	75
13-13	7	(607)	12		3	3	3	1	3	2	2	8	5	9	4	524	6		2	1	19	86	98
14-14	5	(551)	7	7	12	28	27	3	4	8	4	5	2	4	11	7	393	1	13	2	13	71	94
15-15	1	(442)	5	2				2	2		4	1			4		1	415	6			94	100
16-16	2	(946)	6	1		1	1	4	5	8	19	3	7	1	11	2	31	7	834		5	88	99
17-17	17	(47)																		47		100	NC
18-18	18	(248)	2	3	2	5	3		6	5	1	4	1	3	9	29	3		2	1	169	88	69
				192	179	517	247	135	260	285	232	218	93	291	731	556	440	415	840	67	255		
Totals:		7210	428	252	257	627	251	142	280	299	247	229	86	386	778	706	567	431	899	69	266	5572	
Precision:			72	75	70	74	68	84	72	81	68	71	75	71	83	74	69	96	93	68	64		
			NC	98	100	89	69	88	77	85	71	74	77	94	89	94	89	100	99	70	66		

USPTO Class as Training Set

← Classes assigned by Bayesian classifier →

Bayes 0

USPTO

Classes of the test set

USPTO			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Recall	
			19	3	4	6	16	10	11	12	13	14	15	8	9	7	5	1	2	17	18		
0-0	19	(435)	308	8	4	16	1	2	2	4	12	2	1	4	8	29	15	2	5	1	11	71	NC
1-1	3	(230)	2	189		15	1		2		1	6			3	5	5		1			~~82~~	100
2-2	4	(245)	7	2	179	12	1		5		2	2		2	6	11	10		4		2	73	98
3-3	6	(639)	31	23	12	461	14		9	4	2	9		2	23	7	36			1	5	72	89
4-4	16	(211)	2	3		4	171	2	1	2		4	1	1	4	9	3		1	2	1	81	83
5-5	10	(150)			2	4	1	119	2		2			1	4	4	8	1	2			79	83
6-6	11	(235)	3	4	3	6			201			3			4	6			1	1	3	86	90
7-7	12	(343)	4	1	7	10	3	3	9	242	6	8	1	2	14	8	10	2	2	4	7	71	80
8-8	13	(177)	1	1		1					164	2		1	1		2		4			93	94
9-9	14	(184)	2	1		1	3		2		1	162	3	1	1	5			1		1	88	93
10-10	15	(87)								3	1	1	72		2		4		1	3		83	86
11-11	8	(338)	7		2	3			1	1	1			273	20	19	1		1		9	81	92
12-12	9	(1095)	29	7	31	57	22	6	26	20	25	9	3	82	649	41	39	3	19	6	21	59	75
13-13	7	(607)	12		3	3	3	1	3	2	2	8	5	9	4	524	6		2	1	19	86	98
14-14	5	(551)	7	7	12	28	27	3	4	8	4	5	2	4	11	7	393	1	13	2	13	71	94
15-15	1	(442)	5	2				2	2		4	1			4		1	415	6			94	100
16-16	2	(946)	6	1		1	1	4	5	8	19	3	7	1	11	2	31	7	834		5	88	99
17-17	17	(47)																		47		100	NC
18-18	18	(248)	2	3	2	5	3		6	5	1	4	1	3	9	29	3		2	1	169	68	69
Totals:		7210	428	252	257	627	251	142	280	299	247	229	96	386	778	706	567	431	899	69	266	5572	
Precision:			72	75	70	74	68	84	72	81	66	71	75	71	83	74	69	96	93	68	64		

308	229	239	568
+120	+3	+0	+56
428	232	239	624
72%	99%	100%	91%

Can the off diagonals be considered XR's?

ADDITIONAL ANALYSES FOR THE USPTO AS FOLLOWUP TO THE SEPT. 30, 1991 MEETING BETWEEN GLEN SELF AND THE USPTO

EDS Research

5951 Jefferson St. NE

Albuquerque, NM 87109

October 11, 1991

Copyright © 1991, by EDS Research

TABLE OF CONTENTS

Section 1. Introduction 1

Section 2. Cluster analysis of category 12, USPTO Class 5020W 2

Section 3. First half to predict second half 35

Section 1. Introduction.

This paper contains additional analyses for the USPTO as a followup to the September 30, 1991 meeting between Glen Self and the USPTO concerning "Report to the USPTO on the Automatic Classification of Class 395" by EDS Research, September 16, 1991. Section 2 reports the results of hierarchical cluster analysis applied to Category 12, USPTO Class 5020W, Storage Accessing and Control. Section 3 reports the results of two Bayesian categorization tests for Class 395. One test uses the original USPTO classifications. The other uses the Bayes3 classifications. Both tests use the first half of the data to train a Bayesian classifier in order to classify the second half of the data. The overall accuracy was 54.7% for the USPTO data and 74.4% for the Bayes3 data.

Section 2. Cluster Analysis of Category 12, USPTO Class 5020W, Storage Accessing and Control.

This section contains two tables. The first table shows clusters for the 1095 patents that the USPTO put into category 12. The second shows clusters for the 691 patents that Bayes3 put into category 12. The tables resulted from hierarchical cluster analyses of the two data sets. They report the patent number, categories, and title for each of the patents. There are five categories for each patent, one for a 50 category clustering, one for a 40 category clustering, one for a 30 category clustering, one for a 20 category clustering, and one for a 10 category clustering. Because the method is hierarchical, whenever two patents are put into the same cluster, they will be in the same cluster for all clusterings with fewer categories.

After each of the two tables, we give a histogram for each of the five clusterings showing how many patents were put into each category.

We also included two scrolls which show the cluster trees for the 50 category clusterings. Each node in the tree has three numbers. The first is the patent number, the second is the pass in which that node was attached to its parent node, and the third is the distance between that node and its parent on that pass. When two clusters are combined, we use the convention that the cluster with the lowest patent number is the parent of the other cluster.

Cluster Analysis for the USPTO data.

Patent #	50	40	30	20	10	Title
03866180	1	1	1	1	1	Having an instruction pipeline for concurrently processing a plurality of instructions
03883847	1	1	1	1	1	Uniform decoding of minimum-redundancy codes
03883849	1	1	1	1	1	Memory utilizing magnetic bubble domain device
03936803	1	1	1	1	1	Data processing system having a common channel unit with circulating fields
04031516	1	1	1	1	1	Transmission data processing device
04064553	1	1	1	1	1	Information processor
04087853	1	1	1	1	1	Storage reconfiguration apparatus
04103334	1	1	1	1	1	Data handling system involving memory-to-memory transfer
04115850	1	1	1	1	1	Apparatus for performing auxiliary management functions in an associative memory device
04122531	1	1	1	1	1	Memory and control circuit for the memory
04133041	1	1	1	1	1	Data processing control apparatus with selective data readout
04145753	1	1	1	1	1	Comparing apparatus for variable length word
04148098	1	1	1	1	1	Data transfer system with disk command verification apparatus
04150439	1	1	1	1	1	Impression data-processing apparatus
04151598	1	1	1	1	1	Priority assignment apparatus for use in a memory controller
04152762	1	1	1	1	1	Associative crosspoint processor system
04159535	1	1	1	1	1	Framing and elastic store circuit apparatus
04241420	1	1	1	1	1	Disk data control
04371924	1	1	1	1	1	Computer system apparatus for prefetching data requested by a peripheral device from memory
04378594	1	1	1	1	1	High speed to low speed data buffering means
04445191	1	1	1	1	1	Data word handling enhancement in a page oriented named-data hierarchical memory system
04475174	1	1	1	1	1	Decoding apparatus for codes represented by code tree
04601009	1	1	1	1	1	Memory system and accessing method
04636974	1	1	1	1	1	Method and apparatus for transferring data between operationally-juxtaposed memories and knowledge-
04654819	1	1	1	1	1	Memory back-up system
04685057	1	1	1	1	1	Memory mapping system
04718038	1	1	1	1	1	Data security device for storing data at a peripheral part of the device during power down thus preventing
04763251	1	1	1	1	1	Merge and copy bit block transfer implementation
04807179	1	1	1	1	1	Method and device for recording analog parameters on a static digital memory
04914620	1	1	1	1	1	Capacity extensible data storage for use in electronic apparatus
04916658	1	1	1	1	1	Dynamic buffer control
04937779	1	1	1	1	1	Information retrieving apparatus capable of rearranging information stored in memory
04949240	1	1	1	1	1	Data storage system having circuitry for dividing received data into sequential wards each stored in
04967391	1	1	1	1	1	Data string retrieval apparatus for IC card
05003506	1	1	1	1	1	Memory capacity detection apparatus and electronic applied measuring device employing the same
03883854	28	1	1	1	1	Interleaved memory control signal and data handling apparatus using pipelining techniques
04000486	28	1	1	1	1	Full page, raster scan, proportional space character generator
04017838	28	1	1	1	1	Data entry and recording system having field correct capability
04052704	28	1	1	1	1	Apparatus for reordering the sequence of data stored in a serial memory
04054945	28	1	1	1	1	Electronic computer capable of searching a queue in response to a single instruction
04241396	28	1	1	1	1	Tagged pointer handling apparatus
04280177	28	1	1	1	1	Implicit address structure and method for accessing an associative memory device
04285049	28	1	1	1	1	Apparatus and method for selecting finite success states by indexing
04314356	28	1	1	1	1	High-speed term searcher
04322795	28	1	1	1	1	Cache memory utilizing selective clearing and least recently used updating
04334289	28	1	1	1	1	Apparatus for recording the order of usage of locations in memory
04390945	28	1	1	1	1	Self-managing variable field storage station employing a cursor for handling nested data structures
04433392	28	1	1	1	1	Interactive data retrieval apparatus
04486854	28	1	1	1	1	First-in, first-out memory system
04511994	28	1	1	1	1	Multi-group LRU resolver
04574349	28	1	1	1	1	Apparatus for addressing a larger number of instruction addressable central processor registers than can
04607331	28	1	1	1	1	Method and apparatus for implementing an algorithm associated with stored information
04648069	28	1	1	1	1	Character generator
04653022	28	1	1	1	1	Portable electrocardiogram storing apparatus
04700294	28	1	1	1	1	Data storage system having means for compressing input data from sets of correlated parameters
04714990	28	1	1	1	1	Data storage apparatus
04757469	28	1	1	1	1	Method of addressing a random access memory as a delay line, and signal processing device including
04792898	28	1	1	1	1	Method and apparatus for temporarily storing multiple data records
04920483	28	1	1	1	1	A computer memory for accessing any word-sized group of contiguous bits
04937740	28	1	1	1	1	Real time software analyzing system for storing selective m-bit addresses based upon correspondingly
04977498	28	1	1	1	1	Data processing system having a data memory interlock coherency scheme
03868646	21	18	13	9	1	Memory device with standby memory elements
03868649	21	18	13	9	1	MICROPROGRAM CONTROL SYSTEM
03899776	21	18	13	9	1	Programmable terminal
03906453	21	18	13	9	1	Care memory control circuit
03911406	21	18	13	9	1	Correction apparatus for use with a read only memory system
03936805	21	18	13	9	1	Dictation system for storing and retrieving audio information
03965457	21	18	13	9	1	Digital control processor
04008462	21	18	13	9	1	Plural control memory system with multiple micro instruction readout
04044335	21	18	13	9	1	Memory cell output driver
04051460	21	18	13	9	1	Apparatus for accessing an information storage device having defective memory cells
04056844	21	18	13	9	1	Memory control system using plural buffer address arrays
04070703	21	18	13	9	1	Control store organization in a microprogrammed data processing system
04118773	21	18	13	9	1	Microprogram memory bank addressing system
04158227	21	18	13	9	1	Paged memory mapping with elimination of recurrent decoding
04159520	21	18	13	9	1	Memory address control device with extender bus
04168523	21	18	13	9	1	Data processor utilizing a two level microaddressing controller
04175287	21	18	13	9	1	Elastic store slip control circuit apparatus and method for preventing overlapping sequential read and
04247893	21	18	13	9	1	Memory interface device with processing capability
04247905	21	18	13	9	1	Memory clear system
04271484	21	18	13	9	1	Condition code accumulator apparatus for a data processing system
04295205	21	18	13	9	1	Solid state mass memory system compatible with rotating disc memory equipment
04344130	21	18	13	9	1	Apparatus to execute DMA transfer between computing devices using a block move instruction
04346439	21	18	13	9	1	Direct memory access of a main memory by an input/output device using a table in main memory
04396981	21	18	13	9	1	Control store apparatus having dual mode operation handling mechanism

Patent #	50	40	30	20	10	Title
04402041	21	18	13	9	1	Plural storage areas with different priorities in a processor system separated by processor controlled
04426680	21	18	13	9	1	Data processor using read only memories for optimizing main memory access and identifying the starting
04434464	21	18	13	9	1	Memory protection system for effecting alteration of protection information without intervention of control
04435775	21	18	13	9	1	Data processing system having interlinked slow and fast memory means
04446516	21	18	13	9	1	Data compaction system with contiguous storage of non-redundant information and run length counts
04446518	21	18	13	9	1	Microprogrammed control unit with multiple branch capability
04484261	21	18	13	9	1	Data processing system having interlinked fast and slow memory means and interlinked program
04499536	21	18	13	9	1	Signal transfer timing control using stored data relating to operating speeds of memory and processor
04511962	21	18	13	9	1	Memory control unit
04580240	21	18	13	9	1	Memory arrangement operable as a cache and a local memory
04594658	21	18	13	9	1	Hierarchy of control stores for overlapped data transmission
04603384	21	18	13	9	1	Data processing system with multiple memories and program counter
04800491	21	18	13	9	1	Register-stack apparatus
04815033	21	18	13	9	1	Method and apparatus for accessing a color palette synchronously during refreshing of a monitor and
04864491	21	18	13	9	1	Memory device
04901222	21	18	13	9	1	Method and apparatus for backing out of a software instruction after execution has begun
04956804	21	18	13	9	1	Data processing system with memories access time counting and information processor wait signal
03911408	22	18	13	9	1	Apparatus and method for controlling a communications terminal
03940744	22	18	13	9	1	Self contained program loading apparatus
04040034	22	18	13	9	1	Data security system employing automatic time stamping mechanism
04058850	22	18	13	9	1	Programmable controller
04099230	22	18	13	9	1	High level control processor
04135240	22	18	13	9	1	Protection of data file contents
04172282	22	18	13	9	1	Processor controlled memory refresh
04187538	22	18	13	9	1	Read request selection system for redundant storage
04251864	22	18	13	9	1	Apparatus and method in a data processing system for manipulation of signal groups having boundaries
04300192	22	18	13	9	1	Method and means for storing and accessing information in a shared access multiprogrammed data
04355356	22	18	13	9	1	Process and data system using data qualifiers
04377844	22	18	13	9	1	Address translator
04410944	22	18	13	9	1	Apparatus and method for maintaining cache memory integrity in a shared memory environment
04445190	22	18	13	9	1	Program identification encoding
04476528	22	18	13	9	1	Method and apparatus for controlling a data access in a data base management system
04488223	22	18	13	9	1	Control apparatus for a plurality of memory units
04491909	22	18	13	9	1	Data processing system having shared memory
04506323	22	18	13	9	1	Cache/disk file status indicator with data protection feature
04525777	22	18	13	9	1	Split-cycle cache system with SCU controlled cache clearing during cache store access period
04628450	22	18	13	9	1	Data processing system having a local memory which does not use a directory device with distributed
04701844	22	18	13	9	1	Dual cache for independent prefetch and execution units
04734855	22	18	13	9	1	Apparatus and method for fast and stable data storage
04740889	22	18	13	9	1	Cache disable for a data processor
04747039	22	18	13	9	1	Apparatus and method for utilizing an auxiliary data memory unit in a data processing system having
04755936	22	18	13	9	1	Apparatus and method for providing a cache memory unit with a write operation utilizing two system
04783737	22	18	13	9	1	PROM writer adapted to accept new writing algorithm
04799186	22	18	13	9	1	Electronic circuit constituting an improved high-speed stable memory with memory zones protect from
04799635	22	18	13	9	1	System for determining authenticity of an external memory used in an information processing apparatus
04809218	22	18	13	9	1	Apparatus and method for increased system bus utilization in a data processing system
04819204	22	18	13	9	1	Method for controlling memory access on a chip card and apparatus for carrying out the method
04833603	22	18	13	9	1	Apparatus and method for implementation of a page frame replacement algorithm in a data processing
04835682	22	18	13	9	1	Computer system for preventing copying of program from a storage medium by modifying the program
04847804	22	18	13	9	1	Apparatus and method for data copy consistency in a multi-cache data processing unit
04862348	22	18	13	9	1	Microcomputer having high-speed and low-speed operation modes for reading a memory
04896264	22	18	13	9	1	Microprocess with selective cache memory
04912626	22	18	13	9	1	Hit predictive cache memory
04918586	22	18	13	9	1	Extended memory device with instruction read from first control store containing information for
04949238	22	18	13	9	1	Apparatus for detecting memory protection violation
04958276	22	18	13	9	1	Single chip processor
03924245	26	21	16	9	1	Stack mechanism for a data processor
04027291	26	21	16	9	1	Access control unit
04064558	26	21	16	9	1	Method and apparatus for randomizing memory site usage
04166289	26	21	16	9	1	Storage controller for a digital signal processing system
04253147	26	21	16	9	1	Memory unit with pipelined cycle of operations
04293941	26	21	16	9	1	Memory access control system in vector processing system
04442488	26	21	16	9	1	Instruction cache memory system
04502115	26	21	16	9	1	Data processing unit of a microprogram control system for variable length data
04580214	26	21	16	9	1	Memory control system
04615018	26	21	16	9	1	Method for writing data into a memory
04652991	26	21	16	9	1	Data transfer apparatus
04670836	26	21	16	9	1	Device for detecting an overlap of operands to be accessed
04674034	26	21	16	9	1	Data processing machine suitable for high-speed processing
04703449	26	21	16	9	1	Interrupt driven multi-buffer DMA circuit for enabling continuous sequential data transfers
04774687	26	21	16	9	1	Advanced store-in system for a hierarchy memory device
04791564	26	21	16	9	1	Random access memory file apparatus for personal computer with external memory file
04797851	26	21	16	9	1	Data transfer control equipment
04864533	26	21	16	9	1	Data transfer control unit permitting data access to memory prior to completion of data transfer
04866608	26	21	16	9	1	Microprocessor with improved execution of instructions
04881167	26	21	16	9	1	Data memory system
04890226	26	21	16	9	1	Memory access control apparatus having empty real address storing memory and logical address/reat
04937738	26	21	16	9	1	Data processing system which selectively bypasses a cache memory in fetching information based upon
04939636	26	21	16	9	1	Memory management unit
04942521	26	21	16	9	1	Microprocessor with a cache memory in which validity flags for first and second data areas are
05007011	26	21	16	9	1	Data storage device including data address predicting function
05007012	26	21	16	9	1	Fly-by data transfer system
03931613	27	21	16	9	1	Data processing system
03956737	27	21	16	9	1	Memory system with parallel access to multi-word blocks
04048623	27	21	16	9	1	Data processing system

Patent #	50	40	30	20	10	Title
04051551	27	21	16	9	1	Multidimensional parallel access computer memory system
04056845	27	21	16	9	1	Memory access technique
04089052	27	21	16	9	1	Data processing system
04195342	27	21	16	9	1	Multi-configurable cache store system
04197580	27	21	16	9	1	Data processing system including a cache memory
04314332	27	21	16	9	1	Memory control system
04347567	27	21	16	9	1	Computer system apparatus for improving access to memory by deferring write operations
04348720	27	21	16	9	1	Microcomputer arranged for direct memory access
04400774	27	21	16	9	1	Cache addressing arrangement in a computer system
04458310	27	21	16	9	1	Cache memory using a lowest priority replacement circuit
04489395	27	21	16	9	1	Information processor
04493026	27	21	16	9	1	Set associative sector cache
04493033	27	21	16	9	1	Dual port cache with interleaved read accesses during alternate half-cycles and simultaneous writing
04504902	27	21	16	9	1	Cache arrangement for direct memory access block transfer
04514808	27	21	16	9	1	Data transfer system for a data processing system provided with direct memory access units
04638431	27	21	16	9	1	Data processing system for vector processing having a cache invalidation control unit
04646233	27	21	16	9	1	Physical cache unit for computer
04663728	27	21	16	9	1	Read/modify/write circuit for computer memory operation
04733367	27	21	16	9	1	Swap control apparatus for hierarchical memory system
04747042	27	21	16	9	1	Display control system
04783736	27	21	16	9	1	Digital computer with multisection cache
04794521	27	21	16	9	1	Digital computer with cache capable of concurrently handling multiple accesses from parallel processors
04797813	27	21	16	9	1	Cache memory control apparatus
04823259	27	21	16	9	1	High speed buffer store arrangement for quick wide transfer of data
04835678	27	21	16	9	1	Cache memory circuit for processing a read request during transfer of a data block
04853848	27	21	16	9	1	Block access system using cache memory
04884191	27	21	16	9	1	Memory array unit for computer
04905188	27	21	16	9	1	Functional cache memory chip architecture for improved cache access
04910656	27	21	16	9	1	Bus master having selective burst initiation
04912630	27	21	16	9	1	Cache address comparator with sram having burst addressing control
04912631	27	21	16	9	1	Burst mode cache with wrap-around fill
04912632	27	21	16	9	1	Memory control subsystem
04914573	27	21	16	9	1	Bus master which selectively attempts to fill complete entries in a cache line
04920478	27	21	16	9	1	Cache system used in a magnetic disk controller adopting an LRU system
04939641	27	21	16	9	1	Multi-processor system with cache memories
04941088	27	21	16	9	1	Split bus multiprocessing system with data transfer between main memory and caches using interleaving
04994962	27	21	16	9	1	Variable length cache fill
04996641	27	21	16	9	1	Diagnostic mode for a cache
03916384	23	19	14	10	1	Communication switching system computer memory control arrangement
04017840	23	19	14	10	1	Method and apparatus for protecting memory storage location accesses
04267582	23	19	14	10	1	Circuit arrangement for storing a text
04325118	23	19	14	10	1	Instruction fetch circuitry for computers
04332009	23	19	14	10	1	Memory protection system
04395755	23	19	14	10	1	Information processing system and logout process therefor
04453216	23	19	14	10	1	Access control system for a channel buffer
04467419	23	19	14	10	1	Data processing system with access to a buffer store during data block transfers
04527238	23	19	14	10	1	Cache with independent addressable data and directory arrays
04547848	23	19	14	10	1	Access control processing system in computer system
04551799	23	19	14	10	1	Verification of real page numbers of stack stored prefetched instructions from instruction cache
04562532	23	19	14	10	1	Main storage configuration control system
04580217	23	19	14	10	1	High speed memory management system and method
04589064	23	19	14	10	1	System for controlling key storage unit which controls access to main storage
04639862	23	19	14	10	1	Computer system
04701846	23	19	14	10	1	Computer system capable of interruption using special protection code for write interruption region of
04707784	23	19	14	10	1	Prioritized secondary use of a cache with simultaneous access
04722046	23	19	14	10	1	Cache storage priority
04724518	23	19	14	10	1	Odd/even storage in cache memory
04751638	23	19	14	10	1	Buffer storage control system having buffer storage unit comparing operand (OP) side and instruction
04755933	23	19	14	10	1	Data Processor system having look-ahead control
04760546	23	19	14	10	1	Tag control circuit for increasing throughput of main storage access
04783731	23	19	14	10	1	Multicomputer system having dual common memories
04807120	23	19	14	10	1	Temporal garbage collector with indirection cells
04858111	23	19	14	10	1	Write-back cache system using concurrent address transfers to setup requested address in main
04888689	23	19	14	10	1	Apparatus and method for improving cache access throughput in pipelined processors
04897783	23	19	14	10	1	Computer memory system
04916609	23	19	14	10	1	Data processing system for processing units having different throughputs
04953079	23	19	14	10	1	Cache memory address modifier for dynamic alteration of cache block fetch sequence
04954982	23	19	14	10	1	Method and circuit for checking storage protection by pre-checking an access request key
04985825	23	19	14	10	1	System for delaying processing of memory access exceptions until the execution stage of an instruction
03979726	25	19	14	10	1	Apparatus for selectively clearing a cache store in a processor having segmentation and paging
03990051	25	19	14	10	1	Memory steering in a data processing system
04005389	25	19	14	10	1	Arrangement for reducing the access time in a storage system
04051461	25	19	14	10	1	Management table apparatus in memory hierarchy system
04086629	25	19	14	10	1	Hierarchical data store with look-ahead action
04095268	25	19	14	10	1	System for stopping and restarting the operation of a data processor
04130870	25	19	14	10	1	Hierarchially arranged memory system for a data processing arrangement having virtual addressing
04141067	25	19	14	10	1	Multiprocessor system with cache memory
04157586	25	19	14	10	1	Technique for performing partial stores in store-thru memory configuration
04212058	25	19	14	10	1	Computer store mechanism
04298929	25	19	14	10	1	Integrated multilevel storage hierarchy for a data processing system with improved channel to memory
04315312	25	19	14	10	1	Cache memory having a variable data block size
04370710	25	19	14	10	1	Cache memory organization utilizing miss information holding registers to prevent lockup from cache
04380797	25	19	14	10	1	Two level store with many-to-one mapping scheme
04439830	25	19	14	10	1	Computer system key and lock protection mechanism
04449181	25	19	14	10	1	Data processing systems with expanded addressing capability

Patent #	50	40	30	20	10	Title
04462086	25	19	14	10	1	Loading system in numerical controller
04520441	25	19	14	10	1	Data processing system
04550367	25	19	14	10	1	Data processing system having hierarchical memories
04561051	25	19	14	10	1	Memory access method and apparatus in multiple processor systems
04571672	25	19	14	10	1	Access control method for multiprocessor systems
04580211	25	19	14	10	1	Method for controlling storage of data sets in memory unit
04593354	25	19	14	10	1	Disk cache system
04703422	25	19	14	10	1	Memory hierarchy control method with replacement based on request frequency
04729092	25	19	14	10	1	Two store data storage apparatus having a prefetch system for the second store
04761731	25	19	14	10	1	Look-ahead instruction fetch control for a cache memory
04779193	25	19	14	10	1	Data processing apparatus for writing calculation result into buffer memory after the writing of the
04780815	25	19	14	10	1	Memory control method and apparatus
04814981	25	19	14	10	1	Cache invalidate protocol for digital data processing system
04821171	25	19	14	10	1	System of selective purging of address translation in computer memories
04887235	25	19	14	10	1	Symbolic language data processing system
04888686	25	19	14	10	1	System for storing information with comparison of stored data values
04920477	25	19	14	10	1	Virtual address table look aside buffer miss recovery method and apparatus
04984149	25	19	14	10	1	Memory access control apparatus
05008813	25	19	14	10	1	Multi-cache data storage system
03956739	24	20	15	10	1	Data transfer system
04042911	24	20	15	10	1	Outer and asynchronous storage extension system
04047245	24	20	15	10	1	Indirect memory addressing
04075687	24	20	15	10	1	Microprogram controlled digital computer
04079447	24	20	15	10	1	Stored program electronic computer
04122530	24	20	15	10	1	Data management method and system for random access electron beam memory
04156927	24	20	15	10	1	Digital processor system with direct access memory
04218756	24	20	15	10	1	Control circuit for modifying contents of packet switch random access memory
04291388	24	20	15	10	1	Programmable controller with data archive
04296465	24	20	15	10	1	Data mover
04310881	24	20	15	10	1	Conditional transfer control circuit
04349875	24	20	15	10	1	Buffer storage control apparatus
04352165	24	20	15	10	1	Apparatus for storing and retrieving data
04405983	24	20	15	10	1	Auxiliary memory for microprocessor stack overflow
04417304	24	20	15	10	1	Synchronous cycle steal mechanism for transferring data between a processor storage unit and a
04419740	24	20	15	10	1	Method for storing and retrieving data
04426681	24	20	15	10	1	Process and device for managing the conflicts raised by multiple access to same cache memory of a
04439829	24	20	15	10	1	Data processing machine with improved cache memory management
04443865	24	20	15	10	1	Processor module for a programmable controller
04459662	24	20	15	10	1	Microcomputer having ROM mass memory for downloading main RAM memory with microcomputer
04500956	24	20	15	10	1	Memory addressing system
04620279	24	20	15	10	1	Data transfer system
04628477	24	20	15	10	1	Programmable push-pop memory stack
04766535	24	20	15	10	1	High-performance multiple port memory
04821182	24	20	15	10	1	Memory address decoding system
04847750	24	20	15	10	1	Peripheral DMA controller for data acquisition system
04853847	24	20	15	10	1	Data processor with wait control allowing high speed access
04884197	24	20	15	10	1	Method and apparatus for addressing a cache memory
04897780	24	20	15	10	1	Document manager system for allocating storage locations and generating corresponding control blocks
04899275	24	20	15	10	1	Cache-MMU system
04916603	24	20	15	10	1	Distributed reference and change table for a virtual memory system
04928234	24	20	15	10	1	Data processor system and method
04949242	24	20	15	10	1	Microcomputer capable of accessing continuous addresses for a short time
04956805	24	20	15	10	1	Circuitry for character translate functions
04964040	24	20	15	10	1	Computer hardware executive
04050096	49	40	30	20	1	Pulse expanding system for microprocessor systems with slow memory
04223381	49	40	30	20	1	Lookahead memory address control system
04231088	49	40	30	20	1	Allocating and resolving next virtual pages for input/output
04342079	49	40	30	20	1	Duplicated memory system having status indication
04345309	49	40	30	20	1	Relating to cached multiprocessor system with pipeline timing
04357656	49	40	30	20	1	Method and apparatus for disabling and diagnosing cache memory storage locations
04366537	49	40	30	20	1	Authorization mechanism for transfer of program control or data between different address spaces
04426682	49	40	30	20	1	Fast cache flush mechanism
04437149	49	40	30	20	1	Cache memory architecture with decoding
04442487	49	40	30	20	1	Three level memory hierarchy using write and share flags
04445174	49	40	30	20	1	Multiprocessing system including a shared cache
04680703	49	40	30	20	1	Data processing system with reorganization of disk storage for improved paging
04719568	49	40	30	20	1	Hierarchical memory system including separate cache memories for storing data and instructions
04747043	49	40	30	20	1	Multiprocessor cache coherence system
04755930	49	40	30	20	1	Hierarchical cache memory system and method
04807110	49	40	30	20	1	Prefetching system for a cache having a second directory for sequentially accessed blocks
04843542	49	40	30	20	1	Virtual memory cache for use in multi-processing systems
04847758	49	40	30	20	1	Main memory access in a microprocessor system with a cache memory
04885680	49	40	30	20	1	Method and apparatus for efficiently handling temporarily cacheable data
04914582	49	40	30	20	1	Cache tag lookaside
04980823	49	40	30	20	1	Sequential prefetching with deconfirmation
04991088	49	40	30	20	1	Method for optimizing utilization of a cache memory
04158235	50	40	30	20	1	Multi port time-shared associative buffer storage pool
04244033	50	40	30	20	1	Method and system for operating an associative memory
04253144	50	40	30	20	1	Multi-processor communication network
04253146	50	40	30	20	1	Module for coupling computer-processors
04272810	50	40	30	20	1	Arrangement for deleting trailing message portions
04298959	50	40	30	20	1	Digital information transfer system (DITS) receiver
04308580	50	40	30	20	1	Data multiprocessing system having protection against lockout of shared data
04314355	50	40	30	20	1	Apparatus and method for receiving digital data at a first rate and outputting the data at a different rate
04437170	50	40	30	20	1	Method and circuit arrangement for the acceptance and temporary storage of data signals in a switching

Patent #	50	40	30	20	10	Title
04482951	50	40	30	20	1	Direct memory access method for use with a multiplexed data bus
04674033	50	40	30	20	1	Multiprocessor system having a shared memory for enhanced interprocessor communication
04774654	50	40	30	20	1	Apparatus and method for prefetching subblocks from a low speed memory to a high speed memory of a
04775955	50	40	30	20	1	Cache coherence mechanism based on locking
04807122	50	40	30	20	1	Information storage system
04945470	50	40	30	20	1	Hierarchy multi-processor system and control method therefor
04965717	50	40	30	20	1	Multiple processor system having shared memory with private-write capability
04969085	50	40	30	20	1	Memory module for a memory-managed computer system
04972316	50	40	30	20	1	Method of handling disk sector errors in DASD cache
04980818	50	40	30	20	1	Method and apparatus for common resource status management in a computer system including a
03866182	2	2	2	2	2	SYSTEM FOR TRANSFERRING INFORMATION BETWEEN MEMORY BANKS
03934227	2	2	2	2	2	Memory correction system
03976976	2	2	2	2	2	Method and means to access and extended memory unit
03984812	2	2	2	2	2	Computer memory read delay
03987417	2	2	2	2	2	Address memory system
04011544	2	2	2	2	2	Control system having a programmed logic unit
04030078	2	2	2	2	2	Dynamic memory arrangement for providing noncyclic data permutations
04056809	2	2	2	2	2	Fast table lookup apparatus for reading memory
04078259	2	2	2	2	2	Programmable controller having a system for monitoring the logic conditions at external locations
04093982	2	2	2	2	2	Microprocessor system
04171536	2	2	2	2	2	Microprocessor system
04183086	2	2	2	2	2	Computer system having individual computers with data filters
04198683	2	2	2	2	2	Multiple waveform storage system
04215400	2	2	2	2	2	Disk address controller
04225941	2	2	2	2	2	Controller for bubble memories
04245331	2	2	2	2	2	Memory pack
04298934	2	2	2	2	2	Programmable memory protection logic for microprocessor systems
04302809	2	2	2	2	2	External data store memory device
04313159	2	2	2	2	2	Data storage and access apparatus
04344131	2	2	2	2	2	Device for reducing the time of access to information contained in a memory of an information processing
04346441	2	2	2	2	2	Random access memory system for extending the memory addressing capacity of a CPU
04380053	2	2	2	2	2	Memory addressing system for sequentially accessing all memory addresses in a memory area
04382278	2	2	2	2	2	Hierarchial memory system with microcommand memory and pointer register mapping virtual CPU
04424572	2	2	2	2	2	Device for the digital transmission and display of graphics and/or of characters on a screen
04426684	2	2	2	2	2	Scratch pad memory for cassette of magnetic tape recording
04439839	2	2	2	2	2	Dynamically programmable processing element
04456966	2	2	2	2	2	Memory system with flexible replacement units
04459666	2	2	2	2	2	Plural microcode control memory
04473877	2	2	2	2	2	Parasitic memory expansion for computers
04510582	2	2	2	2	2	Binary number substitution mechanism
04513392	2	2	2	2	2	Method and apparatus for generating a repetitive serial pattern using a recirculating shift register
04521852	2	2	2	2	2	Data processing device formed on a single semiconductor substrate having secure memory
04521853	2	2	2	2	2	Secure microprocessor/microcomputer with secured memory
04531199	2	2	2	2	2	Binary number substitution mechanism in a control store element
04570218	2	2	2	2	2	System for the detection of programmable stop codes
04590552	2	2	2	2	2	Security bit for designating the security status of information stored in a nonvolatile memory
04609985	2	2	2	2	2	Microcomputer with severable ROM
04654787	2	2	2	2	2	Apparatus for locating memory modules having different sizes within a memory space
04665480	2	2	2	2	2	Data processing system which permits of using the same erasable and programmable memory for
04706188	2	2	2	2	2	Method and apparatus for reading samples of a time-dependent signal in a data processing system
04713759	2	2	2	2	2	Memory bank switching apparatus
04761729	2	2	2	2	2	Device for exchanging data between a computer and a peripheral unit having a memory formed by shift
04761736	2	2	2	2	2	Memory management unit for addressing an expanded memory in groups of non-contiguous blocks
04777621	2	2	2	2	2	Video game and personal computer
04782439	2	2	2	2	2	Direct memory access system for microcontroller
04783735	2	2	2	2	2	Least recently used replacement level generating apparatus
04792894	2	2	2	2	2	Arithmetic computation modifier based upon data dependent operations for SIMD architectures
04847752	2	2	2	2	2	Data processing apparatus having an input/output controller for controlling interruptions
04847759	2	2	2	2	2	Register selection mechanism and organization of an instruction prefetch buffer
04866603	2	2	2	2	2	Memory control system using a single access request for doubleword data transfers from both odd and
04903231	2	2	2	2	2	Transposition memory for a data processing circuit
04905139	2	2	2	2	2	Cache memory system with improved re-writing address determination scheme involving history of use
04905140	2	2	2	2	2	Semiconductor integrated circuit device
04905142	2	2	2	2	2	Semiconductor integrated circuit device with built-in arrangement for memory testing
04947477	2	2	2	2	2	Partitionable embedded program and data memory for a central processing unit
04984193	2	2	2	2	2	Memory cartridge
03922643	16	14	2	2	2	Memory and memory addressing system
03936802	16	14	2	2	2	Control device for recording elements
04024509	16	14	2	2	2	CCD register array addressing system including apparatus for by-passing selected arrays
04037205	16	14	2	2	2	Digital memory with data manipulation capabilities
04087855	16	14	2	2	2	Valid memory address enable system for a microprocessor system
04115855	16	14	2	2	2	Buffer memory control device having priority control units for priority processing set blocks and unit
04122520	16	14	2	2	2	Microcomputer controller and direct memory access apparatus therefor
04122535	16	14	2	2	2	Storage device
04176403	16	14	2	2	2	Programmable sequence controller
04200928	16	14	2	2	2	Method and apparatus for weighting the priority of access to variable length data blocks in a multiple-disk
04298956	16	14	2	2	2	Digital read recovery with variable frequency compensation using read only memories
04344152	16	14	2	2	2	Buffer memory control circuit for label scanning system
04430724	16	14	2	2	2	Memory interface system having combined command, address and data buss
04488220	16	14	2	2	2	Circuit arrangement for inputting control signals into a microcomputer system
04509142	16	14	2	2	2	Semiconductor memory device with pipeline access
04516218	16	14	2	2	2	Memory system with single command selective sequential accessing of predetermined pluralities of data
04627022	16	14	2	2	2	Pacemaker utilizing microprocessor DMA for generating output pulse sequences
04639890	16	14	2	2	2	Video display system using memory with parallel and serial access employing selectable cascaded serial
04649511	16	14	2	2	2	Dynamic memory controller for single-chip microprocessor

Patent #	50	40	30	20	10	Title
04663735	16	14	2	2	2	Random/serial access mode selection circuit for a video memory system
04689741	16	14	2	2	2	Video system having a dual-port memory with inhibited random access during transfer cycles
04761735	16	14	2	2	2	Data transfer circuit between a processor and a peripheral
04780822	16	14	2	2	2	Semaphore circuit for shared memory cells
04821226	16	14	2	2	2	Dual port video memory system having a bit-serial address input port
04823302	16	14	2	2	2	Block oriented random access memory able to perform a data read, a data write and a data refresh
04875161	16	14	2	2	2	Scientific processor vector file organization
04974156	16	14	2	2	2	Multi-level peripheral data storage hierarchy with independent access to all levels of the hierarchy
RE031977	17	15	10	2	2	Digital computing system having auto-incrementing memory
03972025	17	15	10	2	2	Expanded memory paging for a programmable microprocessor
03974479	17	15	10	2	2	Memory for use in a computer system in which memories have diverse retrieval characteristics
03983537	17	15	10	2	2	Reliability of random access memory systems
03984811	17	15	10	2	2	Memory system with bytewise data transfer control
04040122	17	15	10	2	2	Method and apparatus for refreshing a dynamic memory by sequential transparent readings
04055851	17	15	10	2	2	Memory module with means for generating a control signal that inhibits a subsequent overlapped
04089051	17	15	10	2	2	Alternative direct and indirect addressing
04090238	17	15	10	2	2	Priority vectored interrupt using direct memory access
04095265	17	15	10	2	2	Memory control structure for a pipelined mini-processor system
04099231	17	15	10	2	2	Memory control system for transferring selected words in a multiple memory word exchange during one
04099253	17	15	10	2	2	Random access memory with bit or byte addressing capability
04144564	17	15	10	2	2	Associative memory
04145737	17	15	10	2	2	Associative memory device with time shared comparators
04149262	17	15	10	2	2	Associative memory device with variable recognition criteria
04151593	17	15	10	2	2	Memory module with means for controlling internal timing
04164786	17	15	10	2	2	Apparatus for expanding memory size and direct memory addressing capabilities of digital computer
04164787	17	15	10	2	2	Multiple microprocessor intercommunication arrangement
04204250	17	15	10	2	2	Range count and main memory address accounting system
04214304	17	15	10	2	2	Multiprogrammed data processing system with improved interlock control
04236207	17	15	10	2	2	Memory initialization circuit
04271467	17	15	10	2	2	I/O Priority resolver
04286320	17	15	10	2	2	Digital computing system having auto-incrementing memory
04310879	17	15	10	2	2	Parallel processor having central processor memory extension
04316244	17	15	10	2	2	Memory apparatus for digital computer system
04371928	17	15	10	2	2	Interface for controlling information transfers between main data processing systems units and a central
04374411	17	15	10	2	2	Relocatable read only memory
04383296	17	15	10	2	2	Computer with a memory system for remapping a memory having two memory output buses for high
04387441	17	15	10	2	2	Data processing system wherein at least one subsystem has a local memory and a mailbox memory
04443847	17	15	10	2	2	Page addressing mechanism
04463419	17	15	10	2	2	Microprogram control system
04468729	17	15	10	2	2	Automatic memory module address assignment system for available memory modules
04488231	17	15	10	2	2	Communication multiplexer having dual microprocessors
04489381	17	15	10	2	2	Hierarchical memories having two ports at each subordinate memory level
04507731	17	15	10	2	2	Bidirectional data byte aligner
04513368	17	15	10	2	2	Digital data processing system having object-based logical memory addressing and self-structuring
04519030	17	15	10	2	2	Unique memory for use in a digital data system
04569018	17	15	10	2	2	Digital data processing system having dual-purpose scratchpad and address translation memory
04578751	17	15	10	2	2	System for simultaneously programming a number of EPROMs
04587613	17	15	10	2	2	Microprocessor control system with a bit/byte memory array
04617624	17	15	10	2	2	Multiple configuration memory circuit
04628479	17	15	10	2	2	Terminal with memory write protection
04654788	17	15	10	2	2	Asynchronous multiport parallel access memory system for use in a single board computer system
04658350	17	15	10	2	2	Extended addressing apparatus and method for direct storage access devices
04665481	17	15	10	2	2	Speeding up the response time of the direct multiplex control transfer facility
04665482	17	15	10	2	2	Data multiplex control facility
04691295	17	15	10	2	2	System for storing and retreiving display information in a plurality of memory planes
04694395	17	15	10	2	2	System for performing virtual look-ahead memory operations
04736287	17	15	10	2	2	Set association memory system
04737931	17	15	10	2	2	Memory control device
04740916	17	15	10	2	2	Reconfigurable contiguous address space memory system including serially connected variable capacity
04760522	17	15	10	2	2	Intermixing of different capacity memory array units in a computer
04788640	17	15	10	2	2	Priority logic system
04831522	17	15	10	2	2	Circuit and method for page addressing read only memory
04901234	17	15	10	2	2	Computer system having programmable DMA control
04905137	17	15	10	2	2	Data bus control of ROM units in information processing system
04937781	17	15	10	2	2	Dual port ram with arbitration status register
04980850	17	15	10	2	2	Automatic sizing memory system with multiplexed configuration signals at memory modules
H0000607	3	3	3	3	2	Addressable delay memory
03895360	3	3	3	3	2	Block oriented random access memory
03944989	3	3	3	3	2	Pattern information memory using circulating memories
03949368	3	3	3	3	2	Automatic data priority technique
03949369	3	3	3	3	2	Memory access technique
03958222	3	3	3	3	2	Reconfigurable decoding scheme for memory address signals that uses an associative memory table
03967251	3	3	3	3	2	User variable computer memory module
03971916	3	3	3	3	2	Methods of data storage and data storage systems
04040029	3	3	3	3	2	Memory system with reduced block decoding
04059850	3	3	3	3	2	Memory system word group priority device with least-recently used criterion
04085446	3	3	3	3	2	Data storage and retrieval system
04099258	3	3	3	3	2	System of data storage
04130880	3	3	3	3	2	Data storage system for addressing data stored in adjacent word locations
04174537	3	3	3	3	2	Time-shared, multi-phase memory accessing system having automatically updatable error logging
04277836	3	3	3	3	2	Composite random access memory providing direct and auxiliary memory access
04296475	3	3	3	3	2	Word-organized, content-addressable memory
04306298	3	3	3	3	2	Memory system for microprocessor with multiplexed address/data bus
04318175	3	3	3	3	2	Addressing means for random access memory system
04339804	3	3	3	3	2	Memory system wherein individual bits may be updated

Patent #	50	40	30	20	10	Title
04368515	3	3	3	3	2	Bank switchable memory system
04398248	3	3	3	3	2	Adaptive WSI/MNOS solid state memory system
04425617	3	3	3	3	2	High-speed data sorter
04433388	3	3	3	3	2	Longitudinal parity
04450524	3	3	3	3	2	Single chip microcomputer with external decoder and memory and internal logic for disabling the ROM
04459660	3	3	3	3	2	Microcomputer with automatic refresh of on-chip dynamic RAM transparent to CPU
04472772	3	3	3	3	2	High speed microinstruction execution apparatus
04479180	3	3	3	3	2	Digital memory system utilizing fast and slow address dependent access cycles
04482979	3	3	3	3	2	Video computing system with automatically refreshed memory
04485457	3	3	3	3	2	Memory system including RAM and page switchable ROM
04500962	3	3	3	3	2	Computer system having an extended directly addressable memory space
04506322	3	3	3	3	2	Read/write memory cell for microcomputer
04511966	3	3	3	3	2	Digital signal processing system
04513372	3	3	3	3	2	Universal memory
04513374	3	3	3	3	2	Memory system
04542454	3	3	3	3	2	Apparatus for controlling access to a memory
04546451	3	3	3	3	2	Raster graphics display refresh memory architecture offering rapid access speed
04559611	3	3	3	3	2	Mapping and memory hardware for writing horizontal and vertical lines
04570221	3	3	3	3	2	Apparatus for sorting data words on the basis of the values of associated parameters
04587629	3	3	3	3	2	Random address memory with fast clear
04606003	3	3	3	3	2	Mailing system peripheral interface with replaceable prom for accessing memories
04608632	3	3	3	3	2	Memory paging system in a microcomputer
04609996	3	3	3	3	2	Memory access system for a computer system adapted to accept a memory expansion module
04623986	3	3	3	3	2	Memory access controller having cycle number register for storing the number of column address cycles
04638454	3	3	3	3	2	Digital data storage apparatus
04648032	3	3	3	3	2	Dual purpose screen/memory refresh counter
04654781	3	3	3	3	2	Byte addressable memory for variable length instructions and data
04701843	3	3	3	3	2	Refresh system for a page addressable memory
04725945	3	3	3	3	2	Distributed cache in dynamic rams
04734880	3	3	3	3	2	Dynamic random access memory arrangements having WE, RAS, and CAS derived from a single
04740911	3	3	3	3	2	Dynamically controlled interleaving
04796222	3	3	3	3	2	Memory structure for nonsequential storage of block bytes in multi-bit chips
04797809	3	3	3	3	2	Direct memory access device for multidimensional data transfers
04797850	3	3	3	3	2	Dynamic random access memory controller with multiple independent control channels
04811209	3	3	3	3	2	Cache memory with multiple valid bits for each data indication the validity within different contents
04839856	3	3	3	3	2	Memory access control circuit
04888687	3	3	3	3	2	Memory control system
04908789	3	3	3	3	2	Method and system for automatically assigning memory modules of different predetermined capacities to
04918645	3	3	3	3	2	Computer bus having page mode memory access
04979144	3	3	3	3	2	IC memory card having start address latch and memory capacity output means
04979145	3	3	3	3	2	Structure and method for improving high speed data rate in a DRAM
04037203	7	6	3	3	2	High speed digital information storage system
04128891	7	6	3	3	2	Magnetic bubble domain relational data base system
04161036	7	6	3	3	2	Method and apparatus for random and sequential accessing in dynamic memories
04283771	7	6	3	3	2	On-chip bubble domain relational data base system
04357657	7	6	3	3	2	Floppy-disk interface controller
04449199	7	6	3	3	2	Ultrasound scan conversion and memory system
04563752	7	6	3	3	2	Series/parallel/series shift register memory comprising redundant parallel-connected storage registers,
04602350	7	6	3	3	2	Data reordering memory for use in prime factor transform
04630230	7	6	3	3	2	Solid state storage device
04667308	7	6	3	3	2	Multi-dimensional-access memory system with combined data rotation and multiplexing
04670856	7	6	3	3	2	Data storage apparatus
04751636	7	6	3	3	2	Memory management method and apparatus for initializing and/or clearing R/W storage areas
04787065	7	6	3	3	2	Data processing apparatus providing cyclic addressing of a data store in selectively opposite directions
04847807	7	6	3	3	2	Multiple disk memory access arrangement for gridded type data
04894770	7	6	3	3	2	Set associative memory
04897785	7	6	3	3	2	Search system for locating values in a table using address compare circuit
04914622	7	6	3	3	2	Array-organized bit map with a barrel shifter
04951246	7	6	3	3	2	Nibble-mode dram solid state storage device
04980853	7	6	3	3	2	Bit blitter with narrow shift register
05008852	7	6	3	3	2	Parallel accessible memory device
03944984	4	4	4	4	2	Computer controller system with a reprogrammable read only memory
04030080	4	4	4	4	2	Variable module memory
04068301	4	4	4	4	2	Data storage device comprising search means
04104719	4	4	4	4	2	Multi-access memory module for data processing systems
04128882	4	4	4	4	2	Packet memory system with hierarchical structure
04130885	4	4	4	4	2	Packet memory system for processing many independent memory transactions concurrently
04139901	4	4	4	4	2	Document storage and retrieval system
04156926	4	4	4	4	2	PROM circuit board programmer
04186440	4	4	4	4	2	Continuous memory system
04257097	4	4	4	4	2	Multiprocessor system with demand assignable program paging stores
04287568	4	4	4	4	2	Solid state music player using signals from a bubble-memory storage device
04394734	4	4	4	4	2	Programmable peripheral processing controller
04447875	4	4	4	4	2	Reduction processor for executing programs stored as treelike graphs employing variable-free
04476522	4	4	4	4	2	Programmable peripheral processing controller with mode-selectable address register sequencing
04476541	4	4	4	4	2	Variable function programmed system
04484270	4	4	4	4	2	Centralized hardware control of multisystem access to shared and non-shared subsystems
04498151	4	4	4	4	2	On board non-volatile memory programming
04502127	4	4	4	4	2	Test system memory architecture for passing parameters and testing dynamic components
04519032	4	4	4	4	2	Memory management arrangement for microprocessor systems
04616315	4	4	4	4	2	System memory for a reduction processor evaluating programs stored as binary directed graphs
04652994	4	4	4	4	2	System for transmitting data to auxiliary memory device
04675844	4	4	4	4	2	Ruled line data memory system in a word processing apparatus
04685056	4	4	4	4	2	Computer security device
04725977	4	4	4	4	2	Cartridge programming system and method using a central and local program library

Patent #	50	40	30	20	10	Title
04727509	4	4	4	4	2	Master/slave system for replicating/formatting flexible magnetic diskettes
04787030	4	4	4	4	2	Data processing apparatus with fixed address space
04811275	4	4	4	4	2	Addressing system for an expandable modular electromechanical memory assembly
04852044	4	4	4	4	2	Programmable data security circuit for programmable logic device
04949245	4	4	4	4	2	Intermediate memory system for connecting microcomputers to a rotating disk memory
04956768	4	4	4	4	2	Wideband server, in particular for transmitting music or images
04974152	4	4	4	4	2	Inserted device for independently connecting auxiliary storage units to a data-processing assembly
04980842	4	4	4	4	2	Method and apparatus for registering and retrieving primary information on the basis of secondary
04989137	4	4	4	4	2	Computer memory system
03972028	5	5	5	4	2	Data processing system including a plurality of memory chips each provided with its own address register
04001786	5	5	5	4	2	Automatic configuration of main storage addressing ranges
04005386	5	5	5	4	2	Clearing system
04044338	5	5	5	4	2	Associative memory having separately associable zones
04047160	5	5	5	4	2	Storage arrangement having a device to determine free storage locations
04189767	5	5	5	4	2	Accessing arrangement for interleaved modular memories
04430711	5	5	5	4	2	Central processing unit
04575814	5	5	5	4	2	Programmable interface memory
04677544	5	5	5	4	2	Device for controlling access to a computer memory
04677546	5	5	5	4	2	Guarded regions for controlling memory access
04799149	5	5	5	4	2	Hybrid associative memory composed of a non-associative basic storage and an associative surface, as
04817035	5	5	5	4	2	Method of recording in a disk memory and disk memory system
04914575	5	5	5	4	2	System for transferring data between an interleaved main memory and an I/O device at high speed
04918600	5	5	5	4	2	Dynamic address mapping for conflict-free vector access
04970642	5	5	5	4	2	An apparatus for accessing a memory
04984191	5	5	5	4	2	Limited write non-volatile memory and a franking machine making use thereof
04096571	6	5	5	4	2	System for resolving memory access conflicts among processors and minimizing processor waiting times
04281392	6	5	5	4	2	Memory circuit for programmable machines
04333160	6	5	5	4	2	Memory control system
04571676	6	5	5	4	2	Memory module selection and reconfiguration apparatus in a data processing system
04592011	6	5	5	4	2	Memory mapping method in a data processing system
04744025	6	5	5	4	2	Arrangement for expanding memory capacity
04787060	6	5	5	4	2	Technique for determining maximum physical memory present in a system and for detecting attempts to
04807184	6	5	5	4	2	Modular multiple processor architecture using distributed cross-point switch
04951248	6	5	5	4	2	Self configuring memory system
04970682	6	5	5	4	2	Digital map generator and display system
04982342	6	5	5	4	2	Image processor system having multifunction look-up table units
H0000511	8	7	6	5	3	Data collection system
03918031	8	7	6	5	3	Dual mode bulk memory extension system for a data processing
04067059	8	7	6	5	3	Shared direct memory access controller
04181938	8	7	6	5	3	Processor device
04271466	8	7	6	5	3	Direct memory access control system with byte/word control of data bus
04441162	8	7	6	5	3	Local network interface with control processor ? DMA controller for coupling data processing stations to
04455620	8	7	6	5	3	Direct memory access control apparatus
04467420	8	7	6	5	3	One-chip microcomputer
04481578	8	7	6	5	3	Direct memory access data transfer system for use with plural processors
04538224	8	7	6	5	3	Direct memory access peripheral unit controller
04545014	8	7	6	5	3	Information processing apparatus
04602329	8	7	6	5	3	Data processing system having an address translation unit shared by a CPU and a channel unit
04716523	8	7	6	5	3	Multiple port integrated DMA and interrupt controller and arbitrator
04718006	8	7	6	5	3	Data processor system having improved data throughput in a multiprocessor system
04722051	8	7	6	5	3	Combined read/write cycle for a direct memory access controller
04731737	8	7	6	5	3	High speed intelligent distributed control memory system
04755937	8	7	6	5	3	Method and apparatus for high bandwidth shared memory
04811281	8	7	6	5	3	Work station dealing with image data
04855900	8	7	6	5	3	System for transferring data to a mainframe computer
04878166	8	7	6	5	3	Direct memory access apparatus and methods for transferring data between buses having different
04887224	8	7	6	5	3	Image data processing apparatus capable of high-speed data encoding and/or decoding
04972314	8	7	6	5	3	Data flow signal processor method and apparatus
RE033328	15	7	6	5	3	Write protect control circuit for computer hard disc systems
04047157	15	7	6	5	3	Secondary storage facility for data processing
04054947	15	7	6	5	3	Computer to tape deck interface
04101969	15	7	6	5	3	Secondary storage facility with means for monitoring sector pulses
04144583	15	7	6	5	3	Secondary storage facility with means for monitoring error conditions
04218759	15	7	6	5	3	Sync in-sync out calibration for cable length delays
04229804	15	7	6	5	3	Numerical control unit having a cassette type memory
04342081	15	7	6	5	3	Tape device adapter
04371929	15	7	6	5	3	Multiprocessor system with high density memory set architecture including partitionable cache store
04494215	15	7	6	5	3	Disk system
04525801	15	7	6	5	3	Disk storage controller
04600990	15	7	6	5	3	Apparatus for suspending a reserve operation in a disk drive
04612613	15	7	6	5	3	Digital data bus system for connecting a controller and disk drives
04616338	15	7	6	5	3	FIFO arrangement for temporary data storage
04651277	15	7	6	5	3	Control system for a magnetic disk drive unit
04658356	15	7	6	5	3	Control system for updating a change bit
04734851	15	7	6	5	3	Write protect control circuit for computer hard disc systems
04742448	15	7	6	5	3	Integrated floppy disk drive controller
04773004	15	7	6	5	3	Disk drive apparatus with hierarchical control
04779223	15	7	6	5	3	Display apparatus having an image memory controller utilizing a barrel shifter and a mask controller
04792917	15	7	6	5	3	Control apparatus for simultaneous data transfer
04814977	15	7	6	5	3	Apparatus and method for direct memory to peripheral and peripheral to memory data transfers
05003461	15	7	6	5	3	Cluster controller memory arbiter
05010481	15	7	6	5	3	Main memory control system for virtual machine
03967246	9	8	7	6	3	Digital computer arrangement for communicating data via data buses
04028663	9	8	7	6	3	Digital computer arrangement for high speed memory access
04096572	9	8	7	6	3	Computer system with a memory access arbitrator

Patent #	50	40	30	20	10	Title
04099243	9	8	7	6	3	Memory block protection apparatus
04110823	9	8	7	6	3	Soft display word processing system with multiple autonomous processors
04136386	9	8	7	6	3	Backing store access coordination in a multi-processor system
04180855	9	8	7	6	3	Direct memory access expander unit for use with a microprocessor
04209839	9	8	7	6	3	Shared synchronous memory multiprocessing arrangement
04212057	9	8	7	6	3	Shared memory multi-microprocessor computer system
04313161	9	8	7	6	3	Shared storage for multiple processor systems
04320450	9	8	7	6	3	Protection apparatus for multiple processor systems
04392200	9	8	7	6	3	Cached multiprocessor system with pipeline timing
04400771	9	8	7	6	3	Multi-processor system with programmable memory-access priority control
04449183	9	8	7	6	3	Arbitration scheme for a multiported shared functional device for use in multiprocessing systems
04453214	9	8	7	6	3	Bus arbitrating circuit
04455623	9	8	7	6	3	Apparatus for decreasing current consumption of microprocessors in battery-energized systems
04458313	9	8	7	6	3	Memory access control system
04484262	9	8	7	6	3	Shared memory computer method and apparatus
04549273	9	8	7	6	3	Memory access control circuit
04567562	9	8	7	6	3	Controller for controlling access to a plurality of records that can be accessed and changed by several
04583168	9	8	7	6	3	Read only memory and decode circuit
04590586	9	8	7	6	3	Forced clear of a memory time-out to a maintenance exerciser
04621318	9	8	7	6	3	Multiprocessor system having mutual exclusion control function
04627018	9	8	7	6	3	Priority requestor accelerator
04630197	9	8	7	6	3	Anti-mutilation circuit for protecting dynamic memory
04648065	9	8	7	6	3	Modified snapshot priority enabling two requestors to share a single memory port
04652993	9	8	7	6	3	Multiple output port memory storage module
04675811	9	8	7	6	3	Multi-processor system with hierarchy buffer storages
04698749	9	8	7	6	3	RAM memory overlay gate array circuit
04707781	9	8	7	6	3	Shared memory computer method and apparatus
04733348	9	8	7	6	3	Virtual-memory multiprocessor system for parallel purge operation
04733352	9	8	7	6	3	Lock control for a shared storage in a data processing system
04760521	9	8	7	6	3	Arbitration system using centralized and decentralized arbitrators to access local memories in a multi-
04764865	9	8	7	6	3	Circuit for allocating memory cycles to two processors that share memory
04766537	9	8	7	6	3	Paged memory management unit having stack change control register
04803618	9	8	7	6	3	Multiprocessor system having common memory
04831513	9	8	7	6	3	Memory initialization system
04835672	9	8	7	6	3	Access lock apparatus for use with a high performance storage unit of a digital data processing system
04858107	9	8	7	6	3	Computer device display system using conditionally asynchronous memory accessing by video display
04870572	9	8	7	6	3	Multi-processor system
04891749	9	8	7	6	3	Multiprocessor storage serialization apparatus
04975833	9	8	7	6	3	Multiprocessor system which only allows alternately accessing to shared memory upon receiving read
04984153	9	8	7	6	3	Storage locking control for a plurality of processors which share a common storage unit
04991112	9	8	7	6	3	Graphics system with graphics controller and DRAM controller
03991408	10	9	7	6	3	Self-sequencing memory
04028675	10	9	7	6	3	Method and apparatus for refreshing semiconductor memories in multi-port and multi-module memory
04099256	10	9	7	6	3	Method and apparatus for establishing, reading, and rapidly clearing a translation table memory
04158883	10	9	7	6	3	Refresh control system
04181933	10	9	7	6	3	Memory access and sharing control system
04205373	10	9	7	6	3	System and method for accessing memory connected to different bus and requesting subsystem
04240138	10	9	7	6	3	System for direct access to a memory associated with a microprocessor
04249247	10	9	7	6	3	Refresh system for dynamic RAM memory
04282572	10	9	7	6	3	Multiprocessor memory access system
04293926	10	9	7	6	3	Dynamic type semiconductor memory equipment
04298931	10	9	7	6	3	Character pattern display system
04317169	10	9	7	6	3	Data processing system having centralized memory refresh
04327410	10	9	7	6	3	Processor auto-recovery system
04384326	10	9	7	6	3	Memory security circuit using the simultaneous occurrence of two signals to enable the memory
04485456	10	9	7	6	3	Data processor with R/W memory write inhibit signal
04486834	10	9	7	6	3	Multi-computer system having dual common memory
04488257	10	9	7	6	3	Method for confirming incorporation of a memory into microcomputer system
04489380	10	9	7	6	3	Write protected memory
04493031	10	9	7	6	3	Memory write protection using timers
04509140	10	9	7	6	3	Data transmitting link
04513389	10	9	7	6	3	ROM security circuit
04536839	10	9	7	6	3	Memory request arbitrator
04556952	10	9	7	6	3	Refresh circuit for dynamic memory of a data processor employing a direct memory access controller
04564922	10	9	7	6	3	Postage meter with power-failure resistant memory
04580212	10	9	7	6	3	Computer having correctable read only memory
04586157	10	9	7	6	3	Memory transfer unit
04594656	10	9	7	6	3	Memory refresh apparatus
04611279	10	9	7	6	3	DMA asynchronous mode clock stretch
04625296	10	9	7	6	3	Memory refresh circuit with varying system transparency
04628482	10	9	7	6	3	Common memory control system with two bus masters
04631701	10	9	7	6	3	Dynamic random access memory refresh control system
04635224	10	9	7	6	3	Process and system for transmitting a refresh signal to a semiconductor memory
04727486	10	9	7	6	3	Hardware demand fetch cycle system interface
04729090	10	9	7	6	3	DMA system employing plural bus request and grant signals for improving bus data transfer speed
04755964	10	9	7	6	3	Memory control circuit permitting microcomputer system to utilize static and dynamic rams
04780812	10	9	7	6	3	Common memory system for a plurality of computers
04803653	10	9	7	6	3	Memory control system
04807112	10	9	7	6	3	Microcomputer with a bus accessible from an external apparatus
04821177	10	9	7	6	3	Apparatus for controlling system accesses having multiple command level conditional rotational multiple
04884234	10	9	7	6	3	Dynamic RAM refresh circuit with DMA access
04891752	10	9	7	6	3	Multimode expanded memory space addressing system using independently generated DMA channel
04918650	10	9	7	6	3	Memory control interface apparatus
04965722	10	9	7	6	3	Dynamic memory refresh circuit with a flexible refresh delay dynamic memory
04975832	10	9	7	6	3	Microcomputer system with dual DMA mode transmissions

Patent #	50	40	30	20	10	Title
04977537	10	9	7	6	3	Dram nonvolatizer
03982231	11	10	8	7	4	Prefixing in a multiprocessing system
04115851	11	10	8	7	4	Memory access control system
04137562	11	10	8	7	4	Data acquisition from multiple sources
04298932	11	10	8	7	4	Serial storage subsystem for a data processor
04680730	11	10	8	7	4	Storage control apparatus
04734850	11	10	8	7	4	Data process system including plural storage means each capable of concurrent and intermediate
04809161	11	10	8	7	4	Data storage device
04843543	11	10	8	7	4	Storage control method and apparatus
04870569	11	10	8	7	4	Vector access control system
04910667	11	10	8	7	4	Vector processor with vector buffer memory for read or write of vector data between vector storage and
04916604	11	10	8	7	4	Cache storage apparatus
04949244	11	10	8	7	4	Storage system
04972364	11	10	8	7	4	Memory disk accessing apparatus
04131942	14	13	8	7	4	Non-volatile storage module for a controller
04141068	14	13	8	7	4	Auxiliary ROM memory system
04447887	14	13	8	7	4	Method of rewriting data in non-volatile memory, and system therefor
04578774	14	13	8	7	4	System for limiting access to non-volatile memory in electronic postage meters
04584663	14	13	8	7	4	Apparatus for power-on data integrity check of inputted characters stored in volatile memory
04845662	14	13	8	7	4	Data processor employing run-length coding
04887234	14	13	8	7	4	Portable electronic device with plural memory areas
04901227	14	13	8	7	4	Microcomputer system
04028665	12	11	9	7	4	Information store system comprising a plurality of different shift-registers
04078258	12	11	9	7	4	System for arranging and sharing shift register memory
04084258	12	11	9	7	4	Apparatus for performing multiple operations in a shift register memory
04093986	12	11	9	7	4	Address translation with storage protection
04130868	12	11	9	7	4	Independently controllable multiple address registers for a data processor
04183090	12	11	9	7	4	Magnetic bubble memory equipment
04322812	12	11	9	7	4	Digital data processor providing for monitoring, changing and loading of RAM instruction data
04325116	12	11	9	7	4	Parallel storage access by multiprocessors
04358826	12	11	9	7	4	Apparatus for enabling byte or word addressing of storage organized on a word basis
04376974	12	11	9	7	4	Associative memory system
04388687	12	11	9	7	4	Memory unit
04500958	12	11	9	7	4	Memory controller with data rotation arrangement
04682305	12	11	9	7	4	Storage system
04695947	12	11	9	7	4	Virtual address system having fixed common bus cycles
04712190	12	11	9	7	4	Self-timed random access memory chip
04737933	12	11	9	7	4	CMOS multiport general purpose register
04783732	12	11	9	7	4	Two-wire/three-port RAM for cellular array processor
04821169	12	11	9	7	4	Access verification arrangement for digital data processing system which has demand-paged memory
04894797	12	11	9	7	4	FIFO data storage system using PLA controlled multiplexer for concurrent reading and writing of registers
04949301	12	11	9	7	4	Improved pointer FIFO controller for converting a standard RAM into a simulated dual FIFO by controlling
04958314	12	11	9	7	4	Information recording/reproducing apparatus
04982360	12	11	9	7	4	Memory subsystem
04443860	13	12	9	7	4	System for hi-speed comparisons between variable format input data and stored tabular reference data
04644494	13	12	9	7	4	Solid state memory for aircraft flight data recorder systems
04740894	13	12	9	7	4	Computing processor with memoryless function units each connected to different part of a multiported
04747072	13	12	9	7	4	Pattern addressable memory
04805093	13	12	9	7	4	Content addressable memory
04084234	18	16	11	8	5	Cache write capacity
04084236	18	16	11	8	5	Error detection and correction capability for a memory system
04156906	18	16	11	8	5	Buffer store including control apparatus which facilitates the concurrent processing of a plurality of
04208716	18	16	11	8	5	Cache arrangement for performing simultaneous read/write operations
04245304	18	16	11	8	5	Cache arrangement utilizing a split cycle mode of operation
04268907	18	16	11	8	5	Cache unit bypass apparatus
04312036	18	16	11	8	5	Instruction buffer apparatus of a cache unit
04313158	18	16	11	8	5	Cache apparatus for enabling overlap of instruction fetch operations
04314331	18	16	11	8	5	Cache unit information replacement apparatus
04464717	18	16	11	8	5	Multilevel cache system with graceful degradation capability
04157587	19	16	11	8	5	High speed buffer memory system with word prefetch
04195340	19	16	11	8	5	First in first out activity queue for a cache store
04195341	19	16	11	8	5	Initialization of cache store to assure valid data
04195343	19	16	11	8	5	Round robin replacement for a cache store
04214303	19	16	11	8	5	Word oriented high speed buffer memory system connected to a system bus
04695943	19	16	11	8	5	Multiprocessor shared pipeline cache memory with split cycle and concurrent utilization
04768148	19	16	11	8	5	Read in process memory apparatus
04785395	19	16	11	8	5	Multiprocessor coherent cache system including two level shared cache with separately allocated
04833601	19	16	11	8	5	Cache resiliency in processing a variety of address faults
04884196	19	16	11	8	5	System for providing data for an external circuit and related method
04992930	19	16	11	8	5	Synchronous cache memory system incorporating tie-breaker apparatus for maintaining cache
04319324	20	17	12	8	5	Double word fetch system
04323965	20	17	12	8	5	Sequential chip select decode apparatus and method
04366538	20	17	12	8	5	Memory controller with queue control apparatus
04366539	20	17	12	8	5	Memory controller with burst mode capability
04370712	20	17	12	8	5	Memory controller with address independent burst mode capability
04376972	20	17	12	8	5	Sequential word aligned address apparatus
04410943	20	17	12	8	5	Memory delay start apparatus for a queued memory controller
04451880	20	17	12	8	5	Memory controller with interleaved queuing apparatus
04507730	20	17	12	8	5	Memory system with automatic memory configuration
04545010	20	17	12	8	5	Memory identification apparatus and method
04558429	20	17	12	8	5	Pause apparatus for a memory controller with interleaved queuing apparatus
04731738	20	17	12	8	5	Memory timing and control apparatus
04761730	20	17	12	8	5	Computer memory apparatus
03902163	29	22	17	11	6	Buffered virtual storage and data processing system
04080651	29	22	17	11	6	Memory control processor

Patent #	50	40	30	20	10	Title
04080652	29	22	17	11	6	Data processing system
04126894	29	22	17	11	6	Memory overlay linking system
04456958	29	22	17	11	6	System and method of renaming data items for dependency free code
04471430	29	22	17	11	6	Encachement apparatus
04471431	29	22	17	11	6	Encachement apparatus
04472774	29	22	17	11	6	Encachement apparatus
04473881	29	22	17	11	6	Encachement apparatus
04516203	29	22	17	11	6	Improved apparatus for encaching data whose value does not change during execution of an instruction
04525778	29	22	17	11	6	Computer memory control
04525780	29	22	17	11	6	Data processing system having a memory using object-based information and a protection scheme for
04545012	29	22	17	11	6	Access control system for use in a digital computer system with object-based addressing and call and
04652996	29	22	17	11	6	Encachment apparatus using multiple frames and responding to a key to obtain data therefrom
04656579	29	22	17	11	6	Digital data processing system having a uniquely organized memory system and means for storing and
04670839	29	22	17	11	6	Encachement apparatus using two caches each responsive to a key for simultaneously accessing and
04675810	29	22	17	11	6	Digital data processing system having a uniquely organized memory system using object-based
04817032	29	22	17	11	6	User programmable data processor
04821184	29	22	17	11	6	Universal addressing system for a digital data processing system
04868736	29	22	17	11	6	Code operated access control system for electronic data store
03916385	30	23	18	11	6	Ring checking hardware
03950730	30	23	18	11	6	Apparatus and process for the rapid processing of segmented data
04056850	30	23	18	11	6	Absolute relative position encoder processor and display
04068302	30	23	18	11	6	Computer performance measurement
04121286	30	23	18	11	6	Data processing memory space allocation and deallocation arrangements
04199810	30	23	18	11	6	Radiation hardened register file
04268903	30	23	18	11	6	Stack control system and method for data processor
04320455	30	23	18	11	6	Queue structure for a data processing system
04388695	30	23	18	11	6	Hardware memory write lock circuit
04408274	30	23	18	11	6	Memory protection system using capability registers
04409655	30	23	18	11	6	Hierarchial memory ring protection system using comparisons of requested and previously accessed
04430727	30	23	18	11	6	Storage element reconfiguration
04442484	30	23	18	11	6	Microprocessor memory management and protection mechanism
04490787	30	23	18	11	6	STO Stack control system
04524416	30	23	18	11	6	Stack mechanism with the ability to dynamically alter the size of a stack in a data processing system
04587632	30	23	18	11	6	Lookahead stack oriented computer
04710894	30	23	18	11	6	Access control system for storage having hardware area and software area
04713753	30	23	18	11	6	Secure data processing system architecture with format control
04723223	30	23	18	11	6	Direct memory access controller for reducing access time to transfer information from a disk
04729091	30	23	18	11	6	Directing storage requests prior to address comparator initialization with a reference address range
04758982	30	23	18	11	6	Quasi content addressable memory
04835738	30	23	18	11	6	Register stack for a bit slice processor microsequencer
04942524	30	23	18	11	6	Software trap system which saves previous content of software trap handling pointer in a stack upon
04965718	30	23	18	11	6	Data processing system incorporating a memory resident directive for synchronizing multiple tasks
03916388	31	24	19	12	6	Shifting apparatus for automatic data alignment
04163288	31	24	19	12	6	Associative memory
04363095	31	24	19	12	6	Hit/miss logic for a cache memory
04378591	31	24	19	12	6	Memory management unit for developing multiple physical addresses in parallel for use in a cache
04392201	31	24	19	12	6	Diagnostic subsystem for a cache memory
04424561	31	24	19	12	6	Odd/even bank structure for a cache memory
04445172	31	24	19	12	6	Data steering logic for the output of a cache memory having an odd/even bank structure
04467443	31	24	19	12	6	Bit addressable variable length memory system
04523276	31	24	19	12	6	Input/output control device with memory device for storing variable-length data and method of controlling
04631671	31	24	19	12	6	Data processing system capable of transferring single-byte and double-byte data under DMA control
04658349	31	24	19	12	6	Direct memory access control circuit and data processing system using said circuit
04964037	31	24	19	12	6	Memory addressing arrangement
04086658	34	26	19	12	6	Input/output and diagnostic arrangements for programmable machine controllers having
04128875	34	26	19	12	6	Optional virtual memory system
04156905	34	26	19	12	6	Method and apparatus for improving access speed in a random access memory
04340932	34	26	19	12	6	Dual mapping memory expansion unit
04366536	34	26	19	12	6	Modular digital computer system for storing and selecting data processing procedures and data
04384342	34	26	19	12	6	System for reducing access time to plural memory modules using five present-fetch and one prefetch
04403283	34	26	19	12	6	Extended memory system and method
04502110	34	26	19	12	6	Split-cache having equal size operand and instruction memories
04589092	34	26	19	12	6	Data buffer having separate lock bit storage array
04618926	34	26	19	12	6	Buffer storage control system
04700291	34	26	19	12	6	Memory address control apparatus with separate translation look aside buffers for a data processor using
04703416	34	26	19	12	6	Apparatus for locating programs resident on a cartridge of a cartridge programmable communication
04733350	34	26	19	12	6	Improved purge arrangement for an address translation control system
04737909	34	26	19	12	6	Cache memory address apparatus
04763244	34	26	19	12	6	Paged memory management unit capable of selectively supporting multiple address spaces
04785398	34	26	19	12	6	Virtual cache system using page level number generating CAM to access other memories for processing
04827406	34	26	19	12	6	Memory allocation for multiple processors
04849881	34	26	19	12	6	Data processing unit with a TLB purge function
04875157	34	26	19	12	6	Alternate memory addressing for information storage and retrieval
04942520	34	26	19	12	6	Method and apparatus for indexing, accessing and updating a memory
04943914	34	26	19	12	6	Storage control system in which real address portion of TLB is on same chip as BAA
04985829	34	26	19	12	6	Cache hierarchy design for use in a memory management unit
04991081	34	26	19	12	6	Cache memory addressable by both physical and virtual addresses
04009470	32	25	20	12	6	Pre-emptive, rotational priority system
04070706	32	25	20	12	6	Parallel requestor priority determination and requestor address matching in a cache memory system
04184201	32	25	20	12	6	Integrating processor element
04228503	32	25	20	12	6	Multiplexed directory for dedicated cache memory system
04251863	32	25	20	12	6	Apparatus for correction of memory errors
04393444	32	25	20	12	6	Memory addressing circuit for converting sequential input data to interleaved output data sequence using
04438493	32	25	20	12	6	Multiwork memory data storage and addressing technique and apparatus
04621320	32	25	20	12	6	Multi-user read-ahead memory

Patent #	50	40	30	20	10	Title
04975870	32	25	20	12	6	Apparatus for locking a portion of a computer memory
04381541	33	25	20	12	6	Buffer memory referencing system for two data words
04520439	33	25	20	12	6	Variable field partial write data merge
04600986	33	25	20	12	6	Pipelined split stack with high performance interleaved decode
04649475	33	25	20	12	6	Multiple port memory with port decode error detector
04674032	33	25	20	12	6	High-performance pipelined stack with over-write protection
04697233	33	25	20	12	6	Partial duplication of pipelined stack with data integrity checking
03906455	35	27	21	13	7	Associative memory device
03930234	35	27	21	13	7	Method and apparatus for inserting additional data between data previously stored in a store
04056848	35	27	21	13	7	Memory utilization system
04262332	35	27	21	13	7	Command pair to improve performance and device independence
04310882	35	27	21	13	7	DAS Device command execution sequence
04368513	35	27	21	13	7	Partial roll mode transfer for cyclic bulk memory
04454595	35	27	21	13	7	Buffer for use with a fixed disk controller
04467421	35	27	21	13	7	Virtual storage system and method
04468730	35	27	21	13	7	Detection of sequential data stream for improvements in cache data storage
04476526	35	27	21	13	7	Cache buffered memory subsystem
04489378	35	27	21	13	7	Automatic adjustment of the quantity of prefetch data in a disk cache operation
04490782	35	27	21	13	7	I/O Storage controller cache system with prefetch determined by requested record's position within data
04536836	35	27	21	13	7	Detection of sequential data stream
04573140	35	27	21	13	7	Method of and apparatus for voice communication storage and forwarding with simultaneous access to
04607346	35	27	21	13	7	Apparatus and method for placing data on a partitioned direct access storage device
04773036	35	27	21	13	7	Diskette drive and media type determination
04788641	35	27	21	13	7	Magnetic tape prefetch control system dynamically varying the size of the prefetched block
04807180	35	27	21	13	7	Multiple control system for disk storage and method for realizing same
04811278	35	27	21	13	7	Secondary storage facility employing serial communications between drive and controller
04811280	35	27	21	13	7	Dual mode disk controller
04862411	35	27	21	13	7	Multiple copy data mechanism on synchronous disk drives
04903195	35	27	21	13	7	Method for controlling data transfer
04916605	35	27	21	13	7	Fast write operations
04394732	37	29	21	13	7	Cache/disk subsystem trickle
04394733	37	29	21	13	7	Cache/disk subsystem
04413317	37	29	21	13	7	Multiprocessor system with cache/disk subsystem with status routing for plural disk drives
04415970	37	29	21	13	7	Cache/disk subsystem with load equalization
04419725	37	29	21	13	7	Cache/disk subsystem with tagalong copy
04423479	37	29	21	13	7	Cache/disk subsystem with acquire write command
04425615	37	29	21	13	7	Hierarchical memory system having cache/disk subsystem with command queues for plural disks
04433374	37	29	21	13	7	Cache/disk subsystem with cache bypass
04437155	37	29	21	13	7	Cache/disk subsystem with dual aging of cache entries
04523206	37	29	21	13	7	Cache/disk system with writeback regulation relative to use of cache memory
04523275	37	29	21	13	7	Cache/disk subsystem with floating entry
04530055	37	29	21	13	7	Hierarchical memory system with variable regulation and priority of writeback from cache memory to bulk
04598357	37	29	21	13	7	Cache/disk subsystem with file number for recovery of cached data
04811203	37	29	21	13	7	Hierarchial memory system with separate criteria for replacement and writeback without replacement
04084231	36	28	22	13	7	System for facilitating the copying back of data in disc and tape units of a memory hierarchial system
04189781	36	28	22	13	7	Segmented storage logging and controlling
04197588	36	28	22	13	7	Segmented storage logging and controlling for random entity selection
04198681	36	28	22	13	7	Segmented storage logging and controlling for partial entity selection and condensing
04400793	36	28	22	13	7	Method and arrangement for fast access to CCD-stores
04603380	36	28	22	13	7	DASD cache block staging
04627019	36	28	22	13	7	Database management system for controlling concurrent access to a database
04638424	36	28	22	13	7	Managing data storage devices connected to a digital computer
04648030	36	28	22	13	7	Cache invalidation mechanism for multiprocessor systems
04758944	36	28	22	13	7	Method for managing virtual memory to separate active and stable memory blocks
04758946	36	28	22	13	7	Page mapping system
04771375	36	28	22	13	7	Managing data storage devices connected to a digital computer
04797810	36	28	22	13	7	Incremental, multi-area, generational, copying garbage collector for use in a virtual address space
04833604	36	28	22	13	7	Method for the relocation of linked control blocks
04853842	36	28	22	13	7	Computer memory system having persistent objects
04907151	36	28	22	13	7	System and method for garbage collection with ambiguous roots
04951194	36	28	22	13	7	Method for reducing memory allocations and data copying operations during program calling sequences
04967353	36	28	22	13	7	System for periodically reallocating page frames in memory based upon non-usage within a time period
04974189	36	28	22	13	7	Magnetic tape packet assembler/disassembler safeguards existing data with pretries during appends
04989134	36	28	22	13	7	Method and apparatus for enhancing data storage efficiency
05008786	36	28	22	13	7	Recoverable virtual memory having persistant objects
04414644	38	30	23	14	7	Method and apparatus for discarding data from a buffer after reading such data
04420807	38	30	23	14	7	Selectively holding data in a buffer for defective backing store tracks
04430701	38	30	23	14	7	Method and apparatus for a hierarchical paging storage system
04464713	38	30	23	14	7	Method and apparatus for converting addresses of a backing store having addressable data storage
04499539	38	30	23	14	7	Method and apparatus for limiting allocated data-storage space in a data-storage unit
04574346	38	30	23	14	7	Method and apparatus for peripheral data handling hierarchies
04583166	38	30	23	14	7	Roll mode for cached data storage
04636946	38	30	23	14	7	Method and apparatus for grouping asynchronous recording operations
04638425	38	30	23	14	7	Peripheral data storage having access controls with error recovery
04779189	38	30	23	14	7	Peripheral subsystem initialization method and apparatus
04875155	38	30	23	14	7	Peripheral subsystem having read/write cache with record access
04956803	38	30	23	14	7	Sequentially processing data in a cached data storage system
03911403	39	31	24	15	8	Data storage and processing apparatus
03949377	39	31	24	15	8	Data storage and processing apparatus including processing of spacer characters
03956736	39	31	24	15	8	Disc cartridge sector formatting arrangement and record addressing system
03972027	39	31	24	15	8	Skew compensation for a magnetic card reading-writing unit
03990055	39	31	24	15	8	Control system for magnetic disc storage device
04016547	39	31	24	15	8	Mos shift register compensation system for defective tracks of drum storage system
04019174	39	31	24	15	8	Data collecting and transmitting system
04254472	39	31	24	15	8	Remote metering system

Patent #	50	40	30	20	10	Title
04296477	39	31	24	15	8	Register device for transmission of data having two data ranks one of which receives data only when the
04330824	39	31	24	15	8	Universal arrangement for the exchange of data between the memories and the processing devices of a
04533997	39	31	24	15	8	Computer monitored or controlled system which may be modified and de-bugged on-line by one not
04600999	39	31	24	15	8	Automatic running work vehicle
04780844	39	31	24	15	8	Data input circuit with digital phase locked loop
04845664	39	31	24	15	8	On-chip bit reordering structure
04270185	48	39	24	15	8	Memory control circuitry for a supervisory control system
04500961	48	39	24	15	8	Page mode memory system
04504925	48	39	24	15	8	Self-shifting LIFO stack
04736336	48	39	24	15	8	Asynchronous demand selector with multi-tape delay line
04802086	48	39	24	15	8	FINUFO cache replacement method and apparatus
04860246	48	39	24	15	8	Emulation device for driving a LCD with a CRT display
04907189	48	39	24	15	8	Cache tag comparator with read mode and compare mode
03914747	40	32	25	16	8	Memory having non-fixed relationships between addresses and storage locations
03919694	40	32	25	16	8	Circulating shift register memory having editing and subroutining capability
03924241	40	32	25	16	8	Memory cycle initiation in response to the presence of the memory address
03938097	40	32	25	16	8	Memory and buffer arrangement for digital computers
04065809	40	32	25	16	8	Multi-processing system for controlling microcomputers and memories
04191996	40	32	25	16	8	Self-configurable computer and memory system
04236205	40	32	25	16	8	Access-time reduction control circuit and process for digital storage devices
04244032	40	32	25	16	8	Apparatus for programming a PROM by propagating data words from an address bus to the PROM data
04456953	40	32	25	16	8	Apparatus for and method of addressing data elements in a table having several entries
04495565	40	32	25	16	8	Computer memory address matcher and process
04530049	40	32	25	16	8	Stack cache with fixed size stack frames
04679139	40	32	25	16	8	Method and system for determination of data record order based on keyfield values
04698750	40	32	25	16	8	Security for integrated circuit microcomputer with EEPROM
04736293	40	32	25	16	8	Interleaved set-associative memory
04839796	40	32	25	16	8	Static frame digital memory
04943910	40	32	25	16	8	Memory system compatible with a conventional expanded memory
04498146	41	33	25	16	8	Management of defects in storage media
04577274	41	33	25	16	8	Demand paging scheme for a multi-ATB shared memory processing system
04682284	41	33	25	16	8	Queue administration method and apparatus
04811216	41	33	25	16	8	Multiprocessor memory management method
RE031318	45	37	28	19	8	Automatic modular memory address allocation system
04025903	45	37	28	19	8	Automatic modular memory address allocation system
04028678	45	37	28	19	8	Memory patching circuit
04028679	45	37	28	19	8	Memory patching circuit with increased capability
04028683	45	37	28	19	8	Memory patching circuit with counter
04028684	45	37	28	19	8	Memory patching circuit with repatching capability
04473890	46	38	29	19	8	Method and device for storing stereochemical information about chemical compounds
04791553	46	38	29	19	8	Control unit of input-output interface circuits in an electronic processor
04954951	47	38	29	19	8	System and method for increasing memory performance
03964054	42	34	26	17	9	Hierarchy response priority adjustment mechanism
04152764	42	34	26	17	9	Floating-priority storage control for processors in a multi-processor system
04169284	42	34	26	17	9	Cache control for concurrent access
04189770	42	34	26	17	9	Cache bypass control for operand fetches
04317168	42	34	26	17	9	Cache organization enabling concurrent line castout and line fetch transfers with main storage
04332010	42	34	26	17	9	Cache synonym detection and handling mechanism
04394731	42	34	26	17	9	Cache storage line shareability control for a multiprocessor system
04399506	42	34	26	17	9	Store-in-cache processor means for clearing main storage
04410946	42	34	26	17	9	Cache extension to processor local storage
04463420	42	34	26	17	9	Multiprocessor cache replacement under task control
04464712	42	34	26	17	9	Second level cache replacement method and apparatus
04484267	42	34	26	17	9	Cache sharing control in a multiprocessor
04503497	42	34	26	17	9	System for independent cache-to-cache transfer
04513367	42	34	26	17	9	Cache locking controls in a multiprocessor
04654778	42	34	26	17	9	Direct parallel path for storage accesses unloading common system path
04797814	42	34	26	17	9	Variable address mode cache
04008460	43	35	26	17	9	Circuit for implementing a modified LRU replacement algorithm for a cache
04035779	43	35	26	17	9	Supervisor address key control system
04037214	43	35	26	17	9	Key register controlled accessing system
04037215	43	35	26	17	9	Key controlled address relocation translation system
04038645	43	35	26	17	9	Non-translatable storage protection control system
04050060	43	35	26	17	9	Equate operand address space control system
04084230	43	35	26	17	9	Hybrid semiconductor memory with on-chip associative page addressing, page replacement and control
04093987	43	35	26	17	9	Hardware control storage area protection method and means
04096573	43	35	26	17	9	DLAT Synonym control means for common portions of all address spaces
04142234	43	35	26	17	9	Bias filter memory for filtering out unnecessary interrogations of cache directories in a multiprocessor
04149245	43	35	26	17	9	High speed store request processing control
04280176	43	35	26	17	9	Memory configuration, address interleaving, relocation and access control system
04293910	43	35	26	17	9	Reconfigurable key-in-storage means for protecting interleaved main storage
04472790	43	35	26	17	9	Storage fetch protect override controls
04521846	43	35	26	17	9	Mechanism for accessing multiple virtual address spaces
04581702	43	35	26	17	9	Critical system protection
04068298	44	36	27	18	10	Information storage and retrieval system

USPTO - 10 Clusters

350	uspto_10.1
281	uspto_10.2
135	uspto_10.3
48	uspto_10.4
34	uspto_10.5
94	uspto_10.6
70	uspto_10.7
50	uspto_10.8
32	uspto_10.9
1	uspto_10.10
1095	total

USPTO - 20 Clusters

61	uspto_20.1
141	uspto_20.2
80	uspto_20.3
60	uspto_20.4
46	uspto_20.5
89	uspto_20.6
48	uspto_20.7
34	uspto_20.8
147	uspto_20.9
101	uspto_20.10
44	uspto_20.11
50	uspto_20.12
58	uspto_20.13
12	uspto_20.14
21	uspto_20.15
20	uspto_20.16
32	uspto_20.17
1	uspto_20.18
9	uspto_20.19
41	uspto_20.20
1095	total

USPTO - 30 Clusters

61	uspto_30.1
83	uspto_30.2
80	uspto_30.3
33	uspto_30.4
27	uspto_30.5
46	uspto_30.6
89	uspto_30.7
21	uspto_30.8
27	uspto_30.9
58	uspto_30.10
21	uspto_30.11
13	uspto_30.12
80	uspto_30.13
66	uspto_30.14
35	uspto_30.15
67	uspto_30.16
20	uspto_30.17
24	uspto_30.18
35	uspto_30.19
15	uspto_30.20
37	uspto_30.21
21	uspto_30.22
12	uspto_30.23
21	uspto_30.24
20	uspto_30.25
32	uspto_30.26
1	uspto_30.27
6	uspto_30.28
3	uspto_30.29
41	uspto_30.30
1095	total

USPTO - 40 Clusters

61	uspto_40.1
56	uspto_40.2
60	uspto_40.3
33	uspto_40.4
27	uspto_40.5
20	uspto_40.6
46	uspto_40.7
44	uspto_40.8
45	uspto_40.9
13	uspto_40.10
22	uspto_40.11
5	uspto_40.12
8	uspto_40.13
27	uspto_40.14
58	uspto_40.15
21	uspto_40.16
13	uspto_40.17
80	uspto_40.18
66	uspto_40.19
35	uspto_40.20
67	uspto_40.21
20	uspto_40.22
24	uspto_40.23
12	uspto_40.24
15	uspto_40.25
23	uspto_40.26
23	uspto_40.27
21	uspto_40.28
14	uspto_40.29
12	uspto_40.30
14	uspto_40.31
16	uspto_40.32
4	uspto_40.33
16	uspto_40.34
16	uspto_40.35
1	uspto_40.36
6	uspto_40.37
3	uspto_40.38
7	uspto_40.39
41	uspto_40.40
1095	total

USPTO - 50 Clusters

35	uspto_50.1
56	uspto_50.2
60	uspto_50.3
33	uspto_50.4
16	uspto_50.5
11	uspto_50.6
20	uspto_50.7
22	uspto_50.8
44	uspto_50.9
45	uspto_50.10
13	uspto_50.11
22	uspto_50.12
5	uspto_50.13
8	uspto_50.14
24	uspto_50.15
27	uspto_50.16
58	uspto_50.17
10	uspto_50.18
11	uspto_50.19
13	uspto_50.20
41	uspto_50.21
39	uspto_50.22
31	uspto_50.23
35	uspto_50.24
35	uspto_50.25
26	uspto_50.26
41	uspto_50.27
26	uspto_50.28
20	uspto_50.29
24	uspto_50.30
12	uspto_50.31
9	uspto_50.32
6	uspto_50.33
23	uspto_50.34
23	uspto_50.35
21	uspto_50.36
14	uspto_50.37
12	uspto_50.38
14	uspto_50.39
16	uspto_50.40
4	uspto_50.41
16	uspto_50.42
16	uspto_50.43
1	uspto_50.44
6	uspto_50.45
2	uspto_50.46
1	uspto_50.47
7	uspto_50.48
22	uspto_50.49
19	uspto_50.50
1095	total

Cluster Analysis for the Bayes3 data.

Patent #	50	40	30	20	10	Title
03866180	1	1	1	1	1	Having an instruction pipeline for concurrently processing a plurality of instructions
03956737	1	1	1	1	1	Memory system with parallel access to multi-word blocks
03972028	1	1	1	1	1	Data processing system including a plurality of memory chips each provided with its own address register
03986171	1	1	1	1	1	Storage system comprising a main store and a buffer store
04040034	1	1	1	1	1	Data security system employing automatic time stamping mechanism
04059850	1	1	1	1	1	Memory system word group priority device with least-recently used criterion
04099256	1	1	1	1	1	Method and apparatus for establishing, reading, and rapidly clearing a translation table memory
04099258	1	1	1	1	1	System of data storage
04099259	1	1	1	1	1	Data stores and data storage system
04156905	1	1	1	1	1	Method and apparatus for improving access speed in a random access memory
04344131	1	1	1	1	1	Device for reducing the time of access to information contained in a memory of an information processing
04347567	1	1	1	1	1	Computer system apparatus for improving access to memory by deferring write operations
04467443	1	1	1	1	1	Bit addressable variable length memory system
04504902	1	1	1	1	1	Cache arrangement for direct memory access block transfer
04546451	1	1	1	1	1	Raster graphics display refresh memory architecture offering rapid access speed
04630230	1	1	1	1	1	Solid state storage device
04638454	1	1	1	1	1	Digital data storage apparatus
04646236	1	1	1	1	1	Pipelined control apparatus with multi-process address storage
04796222	1	1	1	1	1	Memory structure for nonsequential storage of block bytes in multi-bit chips
03943347	36	1	1	1	1	Data processor reorder random access memory
03976980	36	1	1	1	1	Data reordering system
03984811	36	1	1	1	1	Memory system with bytewise data transfer control
04040029	36	1	1	1	1	Memory system with reduced block decoding
04058850	36	1	1	1	1	Programmable controller
04070703	36	1	1	1	1	Control store organization in a microprogrammed data processing system
04145753	36	1	1	1	1	Comparing apparatus for variable length word
04158227	36	1	1	1	1	Paged memory mapping with elimination of recurrent decoding
04163288	36	1	1	1	1	Associative memory
04198683	36	1	1	1	1	Multiple waveform storage system
04262338	36	1	1	1	1	Display system with two-level memory control for display units
04277836	36	1	1	1	1	Composite random access memory providing direct and auxiliary memory access
04334287	36	1	1	1	1	Buffer memory arrangement
04370733	36	1	1	1	1	Pattern generation system
04380053	36	1	1	1	1	Memory addressing system for sequentially accessing all memory addresses in a memory area
04425617	36	1	1	1	1	High-speed data sorter
04429367	36	1	1	1	1	Speech synthesizer apparatus
04434464	36	1	1	1	1	Memory protection system for effecting alteration of protection information without intervention of control
04435775	36	1	1	1	1	Data processing system having interlinked slow and fast memory means
04475176	36	1	1	1	1	Memory control system
04500962	36	1	1	1	1	Computer system having an extended directly addressable memory space
04535420	36	1	1	1	1	Circular-queue structure
04538241	36	1	1	1	1	Address translation buffer
04613944	36	1	1	1	1	Electronic translator having removable data memory and controller both connectable to any one of
04620277	36	1	1	1	1	Multimaster CPU system with early memory addressing
04769767	36	1	1	1	1	Memory patching system
04870572	36	1	1	1	1	Multi-processor system
04875157	36	1	1	1	1	Alternate memory addressing for information storage and retrieval
04897783	36	1	1	1	1	Computer memory system
04905139	36	1	1	1	1	Cache memory system with improved re-writing address determination scheme involving history of use
04916603	36	1	1	1	1	Distributed reference and change table for a virtual memory system
03895360	35	29	1	1	1	Block oriented random access memory
04078258	35	29	1	1	1	System for arranging and sharing shift register memory
04084258	35	29	1	1	1	Apparatus for performing multiple operations in a shift register memory
04122531	35	29	1	1	1	Memory and control circuit for the memory
04149242	35	29	1	1	1	Data interface apparatus for multiple sequential processors
04199810	35	29	1	1	1	Radiation hardened register file
04282572	35	29	1	1	1	Multiprocessor memory access system
04388687	35	29	1	1	1	Memory unit
04562534	35	29	1	1	1	Data processing system having a control device for controlling an intermediate memory during a bulk
04594656	35	29	1	1	1	Memory refresh apparatus
04618936	35	29	1	1	1	Synthetic speech speed control in an electronic cash register
04675840	35	29	1	1	1	Speech processor system with auxiliary memory access
04783732	35	29	1	1	1	Two-wire/three-port RAM for cellular array processor
04835684	35	29	1	1	1	Microcomputer capable of transferring data from one location to another within a memory without an
04894797	35	29	1	1	1	FIFO data storage system using PLA controlled multiplexer for concurrent reading and writing of registers
03883849	33	27	21	14	1	Memory utilizing magnetic bubble domain device
03987419	33	27	21	14	1	High speed information processing system
04016547	33	27	21	14	1	Mos shift register compensation system for defective tracks of drum storage system
04028665	33	27	21	14	1	Information store system comprising a plurality of different shift-registers
04086629	33	27	21	14	1	Hierarchical data store with look-ahead action
04124892	33	27	21	14	1	Data processing systems
04128891	33	27	21	14	1	Magnetic bubble domain relational data base system
04137562	33	27	21	14	1	Data acquisition from multiple sources
04138720	33	27	21	14	1	Time-shared, multi-phase memory accessing system
04148098	33	27	21	14	1	Data transfer system with disk command verification apparatus
04159535	33	27	21	14	1	Framing and elastic store circuit apparatus
04171536	33	27	21	14	1	Microprocessor system
04212058	33	27	21	14	1	Computer store mechanism
04322815	33	27	21	14	1	Hierarchical data storage system
04330825	33	27	21	14	1	Device for automatic control of the storage capacity put to work in data processing systems
04349875	33	27	21	14	1	Buffer storage control apparatus
04380797	33	27	21	14	1	Two level store with many-to-one mapping scheme
04445191	33	27	21	14	1	Data word handling enhancement in a page oriented named-data hierarchical memory system
04490787	33	27	21	14	1	STO Stack control system
04561051	33	27	21	14	1	Memory access method and apparatus in multiple processor systems

Patent #	50	40	30	20	10	Title
04570221	33	27	21	14	1	Apparatus for sorting data words on the basis of the values of associated parameters
04584642	33	27	21	14	1	Logic simulation apparatus
04714990	33	27	21	14	1	Data storage apparatus
04729092	33	27	21	14	1	Two store data storage apparatus having a prefetch system for the second store
04763251	33	27	21	14	1	Merge and copy bit block transfer implementation
04766537	33	27	21	14	1	Paged memory management unit having stack change control register
04787065	33	27	21	14	1	Data processing apparatus providing cyclic addressing of a data store in selectively opposite directions
04803655	33	27	21	14	1	Data processing system employing a plurality of rapidly switchable pages for providing data transfer
04835738	33	27	21	14	1	Register stack for a bit slice processor microsequencer
04894770	33	27	21	14	1	Set associative memory
04914620	33	27	21	14	1	Capacity extensible data storage for use in electronic apparatus
05007015	33	27	21	14	1	Portable compact device
04064553	34	28	21	14	1	Information processor
04087853	34	28	21	14	1	Storage reconfiguration apparatus
04103334	34	28	21	14	1	Data handling system involving memory-to-memory transfer
04130880	34	28	21	14	1	Data storage system for addressing data stored in adjacent word locations
04133041	34	28	21	14	1	Data processing control apparatus with selective data readout
04152762	34	28	21	14	1	Associative crosspoint processor system
04184201	34	28	21	14	1	Integrating processor element
04227244	34	28	21	14	1	Closed loop address
04251863	34	28	21	14	1	Apparatus for correction of memory errors
04376974	34	28	21	14	1	Associative memory system
04570222	34	28	21	14	1	Information processor having information correcting function
04601009	34	28	21	14	1	Memory system and accessing method
04949240	34	28	21	14	1	Data storage system having circuitry for dividing received data into sequential wards each stored in
05003506	34	28	21	14	1	Memory capacity detection apparatus and electronic applied measuring device employing the same
03971916	43	35	26	18	1	Methods of data storage and data storage systems
04030080	43	35	26	18	1	Variable module memory
04037205	43	35	26	18	1	Digital memory with data manipulation capabilities
04156926	43	35	26	18	1	PROM circuit board programmer
04183090	43	35	26	18	1	Magnetic bubble memory equipment
04498151	43	35	26	18	1	On board non-volatile memory programming
04652991	43	35	26	18	1	Data transfer apparatus
04685056	43	35	26	18	1	Computer security device
04718038	43	35	26	18	1	Data security device for storing data at a peripheral part of the device during power down thus preventing
04783735	43	35	26	18	1	Least recently used replacement level generating apparatus
04792917	43	35	26	18	1	Control apparatus for simultaneous data transfer
04852044	43	35	26	18	1	Programmable data security circuit for programmable logic device
04937779	43	35	26	18	1	Information retrieving apparatus capable of rearranging information stored in memory
04779223	46	35	26	18	1	Display apparatus having an image memory controller utilizing a barrel shifter and a mask controller
04845662	46	35	26	18	1	Data processor employing run-length coding
04158235	44	36	27	18	1	Multi port time-shared associative buffer storage pool
04747072	44	36	27	18	1	Pattern addressable memory
04805093	44	36	27	18	1	Content addressable memory
04969085	44	36	27	18	1	Memory module for a memory-managed computer system
04644494	45	36	27	18	1	Solid state memory for aircraft flight data recorder systems
03866182	2	2	2	2	2	SYSTEM FOR TRANSFERRING INFORMATION BETWEEN MEMORY BANKS
04051460	2	2	2	2	2	Apparatus for accessing an information storage device having defective memory cells
04085446	2	2	2	2	2	Data storage and retrieval system
04086658	2	2	2	2	2	Input/output and diagnostic arrangements for programmable machine controllers having
04298934	2	2	2	2	2	Programmable memory protection logic for microprocessor systems
04383296	2	2	2	2	2	Computer with a memory system for remapping a memory having two memory output buses for high
04417318	2	2	2	2	2	Arrangement for control of the operation of a random access memory in a data processing system
04456971	2	2	2	2	2	Semiconductor RAM that is accessible in magnetic disc storage format
04458309	2	2	2	2	2	Apparatus for loading programmable hit matrices used in a hardware monitoring interface unit
04503491	2	2	2	2	2	Computer with expanded addressing capability
04511966	2	2	2	2	2	Digital signal processing system
04521849	2	2	2	2	2	Programmable hit matrices used in a hardware monitoring interface unit
04587613	2	2	2	2	2	Microprocessor control system with a bit/byte memory array
04598359	2	2	2	2	2	Apparatus for forward or reverse reading of multiple variable length operands
04617624	2	2	2	2	2	Multiple configuration memory circuit
04631699	2	2	2	2	2	Firmware simulation of diskette data via a video signal
04652994	2	2	2	2	2	System for transmitting data to auxiliary memory device
04656597	2	2	2	2	2	Video system controller with a row address override circuit
04691295	2	2	2	2	2	System for storing and retreiving display information in a plurality of memory planes
04713759	2	2	2	2	2	Memory bank switching apparatus
04736287	2	2	2	2	2	Set association memory system
04740916	2	2	2	2	2	Reconfigurable contiguous address space memory system including serially connected variable capacity
04761736	2	2	2	2	2	Memory management unit for addressing an expanded memory in groups of non-contiguous blocks
04771402	2	2	2	2	2	Address comparator
04777621	2	2	2	2	2	Video game and personal computer
04792891	2	2	2	2	2	Data processor
04802119	2	2	2	2	2	Single chip microcomputer with patching and configuration controlled by on-board non-volatile memory
04839856	2	2	2	2	2	Memory access control circuit
04866603	2	2	2	2	2	Memory control system using a single access request for doubleword data transfers from both odd and
04905140	2	2	2	2	2	Semiconductor integrated circuit device
04937782	2	2	2	2	2	Counter control method
04953101	2	2	2	2	2	Software configurable memory architecture for data processing system having graphics capability
04959777	2	2	2	2	2	Write-shared cache circuit for multiprocessor system
04974143	2	2	2	2	2	Information processing apparatus wherein address path is used for continuous data transfer
04979144	2	2	2	2	2	IC memory card having start address latch and memory capacity output means
04984193	2	2	2	2	2	Memory cartridge
04991110	2	2	2	2	2	Graphics processor with staggered memory timing
03924241	7	6	5	2	2	Memory cycle initiation in response to the presence of the memory address
03958222	7	6	5	2	2	Reconfigurable decoding scheme for memory address signals that uses an associative memory table

Patent #	50	40	30	20	10	Title
03967251	7	6	5	2	2	User variable computer memory module
04065809	7	6	5	2	2	Multi-processing system for controlling microcomputers and memories
04093985	7	6	5	2	2	Memory sparing arrangement
04191996	7	6	5	2	2	Self-configurable computer and memory system
04281392	7	6	5	2	2	Memory circuit for programmable machines
04330842	7	6	5	2	2	Valid memory address pin elimination
04333142	7	6	5	2	2	Self-configurable computer and memory system
04334289	7	6	5	2	2	Apparatus for recording the order of usage of locations in memory
04368515	7	6	5	2	2	Bank switchable memory system
04393443	7	6	5	2	2	Memory mapping system
04446533	7	6	5	2	2	Stored program digital data processor
04473877	7	6	5	2	2	Parasitic memory expansion for computers
04479180	7	6	5	2	2	Digital memory system utilizing fast and slow address dependent access cycles
04485457	7	6	5	2	2	Memory system including RAM and page switchable ROM
04513374	7	6	5	2	2	Memory system
04587629	7	6	5	2	2	Random address memory with fast clear
04608632	7	6	5	2	2	Memory paging system in a microcomputer
04609996	7	6	5	2	2	Memory access system for a computer system adapted to accept a memory expansion module
04613953	7	6	5	2	2	Paging register for memory devices
04742474	7	6	5	2	2	Variable access frame buffer memory
04764896	7	6	5	2	2	Microprocessor assisted memory to memory move apparatus
04835736	7	6	5	2	2	Data acquisition system for capturing and storing clustered test data occurring before and after an event
04860252	7	6	5	2	2	Self-adaptive computer memory address allocation system
04888687	7	6	5	2	2	Memory control system
04908789	7	6	5	2	2	Method and system for automatically assigning memory modules of different predetermined capacities to
04920483	7	6	5	2	2	A computer memory for accessing any word-sized group of contiguous bits
04943910	7	6	5	2	2	Memory system compatible with a conventional expanded memory
04979145	7	6	5	2	2	Structure and method for improving high speed data rate in a DRAM
04996641	7	6	5	2	2	Diagnostic mode for a cache
04999805	7	6	5	2	2	Extended input/output circuit board addressing system
H0000607	8	7	5	2	2	Addressable delay memory
04028675	8	7	5	2	2	Method and apparatus for refreshing semiconductor memories in multi-port and multi-module memory
04287568	8	7	5	2	2	Solid state music player using signals from a bubble-memory storage device
04398248	8	7	5	2	2	Adaptive WSI/MNOS solid state memory system
04482979	8	7	5	2	2	Video computing system with automatically refreshed memory
04513372	8	7	5	2	2	Universal memory
04542454	8	7	5	2	2	Apparatus for controlling access to a memory
04575814	8	7	5	2	2	Programmable interface memory
04623986	8	7	5	2	2	Memory access controller having cycle number register for storing the number of column address cycles
04648032	8	7	5	2	2	Dual purpose screen/memory refresh counter
04701843	8	7	5	2	2	Refresh system for a page addressable memory
04725945	8	7	5	2	2	Distributed cache in dynamic rams
04797850	8	7	5	2	2	Dynamic random access memory controller with multiple independent control channels
04858107	8	7	5	2	2	Computer device display system using conditionally asynchronous memory accessing by video display
03934227	9	8	6	2	2	Memory correction system
03983537	9	8	6	2	2	Reliability of random access memory systems
04001786	9	8	6	2	2	Automatic configuration of main storage addressing ranges
04016545	9	8	6	2	2	Plural memory controller apparatus
04069511	9	8	6	2	2	Digital bit image memory system
04099253	9	8	6	2	2	Random access memory with bit or byte addressing capability
04268901	9	8	6	2	2	Variable configuration accounting machine with automatic identification of the number and type of
04286321	9	8	6	2	2	Common bus communication system in which the width of the address field is greater than the number of
04325116	9	8	6	2	2	Parallel storage access by multiprocessors
04342079	9	8	6	2	2	Duplicated memory system having status indication
04456966	9	8	6	2	2	Memory system with flexible replacement units
04481579	9	8	6	2	2	Digital data apparatus having a plurality of selectively addressable peripheral units
04507731	9	8	6	2	2	Bidirectional data byte aligner
04511960	9	8	6	2	2	Data processing system auto address development logic for multiword fetch
04511961	9	8	6	2	2	Apparatus for measuring program execution
04571676	9	8	6	2	2	Memory module selection and reconfiguration apparatus in a data processing system
04592011	9	8	6	2	2	Memory mapping method in a data processing system
04631671	9	8	6	2	2	Data processing system capable of transferring single-byte and double-byte data under DMA control
04663728	9	8	6	2	2	Read/modify/write circuit for computer memory operation
04688191	9	8	6	2	2	Single bit storage and retrieval with transition intelligence
04760522	9	8	6	2	2	Intermixing of different capacity memory array units in a computer
04783736	9	8	6	2	2	Digital computer with multisection cache
04787060	9	8	6	2	2	Technique for determining maximum physical memory present in a system and for detecting attempts to
04794521	9	8	6	2	2	Digital computer with cache capable of concurrently handling multiple accesses from parallel processors
04847807	9	8	6	2	2	Multiple disk memory access arrangement for gridded type data
04888686	9	8	6	2	2	System for storing information with comparison of stored data values
04896263	9	8	6	2	2	Multi-microcomputer system
04918587	9	8	6	2	2	Prefetch circuit for a computer memory subject to consecutive addressing
04937736	9	8	6	2	2	Memory controller for protected memory with automatic access granting capability
04949245	9	8	6	2	2	Intermediate memory system for connecting microcomputers to a rotating disk memory
04951248	9	8	6	2	2	Self configuring memory system
04964037	9	8	6	2	2	Memory addressing arrangement
03906453	3	3	3	3	2	Care memory control circuit
03911408	3	3	3	3	2	Apparatus and method for controlling a communications terminal
03949368	3	3	3	3	2	Automatic data priority technique
03949369	3	3	3	3	2	Memory access technique
03984812	3	3	3	3	2	Computer memory read delay
04045781	3	3	3	3	2	Memory module with selectable byte addressing for digital data processing system
04047245	3	3	3	3	2	Indirect memory addressing
04055851	3	3	3	3	2	Memory module with means for generating a control signal that inhibits a subsequent overlapped
04095265	3	3	3	3	2	Memory control structure for a pipelined mini-processor system

Patent #	50	40	30	20	10	Title
04144564	3	3	3	3	2	Associative memory
04145737	3	3	3	3	2	Associative memory device with time shared comparators
04149262	3	3	3	3	2	Associative memory device with variable recognition criteria
04151593	3	3	3	3	2	Memory module with means for controlling internal timing
04181933	3	3	3	3	2	Memory access and sharing control system
04183086	3	3	3	3	2	Computer system having individual computers with data filters
04229804	3	3	3	3	2	Numerical control unit having a cassette type memory
04236207	3	3	3	3	2	Memory initialization circuit
04388695	3	3	3	3	2	Hardware memory write lock circuit
04405983	3	3	3	3	2	Auxiliary memory for microprocessor stack overflow
04488222	3	3	3	3	2	Circuit for reusing previously fetched data
04500958	3	3	3	3	2	Memory controller with data rotation arrangement
04616315	3	3	3	3	2	System memory for a reduction processor evaluating programs stored as binary directed graphs
04628477	3	3	3	3	2	Programmable push-pop memory stack
04761731	3	3	3	3	2	Look-ahead instruction fetch control for a cache memory
04783731	3	3	3	3	2	Multicomputer system having dual common memories
04821169	3	3	3	3	2	Access verification arrangement for digital data processing system which has demand-paged memory
04870562	3	3	3	3	2	Microcomputer capable of accessing internal memory at a desired variable access time
04896262	3	3	3	3	2	Emulation device for converting magnetic disc memory mode signal from computer into semiconductor
04982360	3	3	3	3	2	Memory subsystem
04080651	6	5	3	3	2	Memory control processor
04080652	6	5	3	3	2	Data processing system
04099231	6	5	3	3	2	Memory control system for transferring selected words in a multiple memory word exchange during one
04130868	6	5	3	3	2	Independently controllable multiple address registers for a data processor
04172282	6	5	3	3	2	Processor controlled memory refresh
04317169	6	5	3	3	2	Data processing system having centralized memory refresh
04332008	6	5	3	3	2	Microprocessor apparatus and method
04346441	6	5	3	3	2	Random access memory system for extending the memory addressing capacity of a CPU
04354225	6	5	3	3	2	Intelligent main store for data processing systems
04450524	6	5	3	3	2	Single chip microcomputer with external decoder and memory and internal logic for disabling the ROM
04459660	6	5	3	3	2	Microcomputer with automatic refresh of on-chip dynamic RAM transparent to CPU
04516218	6	5	3	3	2	Memory system with single command selective sequential accessing of predetermined pluralities of data
04521852	6	5	3	3	2	Data processing device formed on a single semiconductor substrate having secure memory
04521853	6	5	3	3	2	Secure microprocessor/microcomputer with secured memory
04542453	6	5	3	3	2	Program patching in microcomputer
04586133	6	5	3	3	2	Multilevel controller for a cache memory interface in a multiprocessing system
04590552	6	5	3	3	2	Security bit for designating the security status of information stored in a nonvolatile memory
04628482	6	5	3	3	2	Common memory control system with two bus masters
04654782	6	5	3	3	2	Variable segment size plural cache system with cache memory unit selection based on relative priorities
04665481	6	5	3	3	2	Speeding up the response time of the direct multiplex control transfer facility
04665482	6	5	3	3	2	Data multiplex control facility
04698749	6	5	3	3	2	RAM memory overlay gate array circuit
04720812	6	5	3	3	2	High speed program store with bootstrap
04758982	6	5	3	3	2	Quasi content addressable memory
04811202	6	5	3	3	2	Quadruply extended time multiplexed information bus for reducing the ?pin out? configuration of a
04947477	6	5	3	3	2	Partitionable embedded program and data memory for a central processing unit
04961136	6	5	3	3	2	Microprocessor system having cache directory and cache memory and hardware for initializing the
04040122	4	4	4	3	2	Method and apparatus for refreshing a dynamic memory by sequential transparent readings
04048623	4	4	4	3	2	Data processing system
04056845	4	4	4	3	2	Memory access technique
04089052	4	4	4	3	2	Data processing system
04410944	4	4	4	3	2	Apparatus and method for maintaining cache memory integrity in a shared memory environment
04525777	4	4	4	3	2	Split-cycle cache system with SCU controlled cache clearing during cache store access period
04744025	4	4	4	3	2	Arrangement for expanding memory capacity
04747039	4	4	4	3	2	Apparatus and method for utilizing an auxiliary data memory unit in a data processing system having
04755936	4	4	4	3	2	Apparatus and method for providing a cache memory unit with a write operation utilizing two system
04809218	4	4	4	3	2	Apparatus and method for increased system bus utilization in a data processing system
04847804	4	4	4	3	2	Apparatus and method for data copy consistency in a multi-cache data processing unit
04941088	4	4	4	3	2	Split bus multiprocessing system with data transfer between main memory and caches using interleaving
04954946	4	4	4	3	2	Apparatus and method for providing distribution control in a main memory unit of a data processing
04980850	4	4	4	3	2	Automatic sizing memory system with multiplexed configuration signals at memory modules
04982322	4	4	4	3	2	Apparatus and method for prohibiting access in a multi-cache data processing system to data signal
RE031977	5	4	4	3	2	Digital computing system having auto-incrementing memory
04141068	5	4	4	3	2	Auxiliary ROM memory system
04286320	5	4	4	3	2	Digital computing system having auto-incrementing memory
04468729	5	4	4	3	2	Automatic memory module address assignment system for available memory modules
03866183	10	9	7	4	3	Communications control apparatus for the use with a cache store
03979726	10	9	7	4	3	Apparatus for selectively clearing a cache store in a processor having segmentation and paging
04241396	10	9	7	4	3	Tagged pointer handling apparatus
04257097	10	9	7	4	3	Multiprocessor system with demand assignable program paging stores
04264953	10	9	7	4	3	Virtual cache
04322795	10	9	7	4	3	Cache memory utilizing selective clearing and least recently used updating
04345309	10	9	7	4	3	Relating to cached multiprocessor system with pipeline timing
04349871	10	9	7	4	3	Duplicate tag store for cached multiprocessor system
04370710	10	9	7	4	3	Cache memory organization utilizing miss information holding registers to prevent lockup from cache
04392200	10	9	7	4	3	Cached multiprocessor system with pipeline timing
04471429	10	9	7	4	3	Apparatus for cache clearing
04527238	10	9	7	4	3	Cache with independent addressable data and directory arrays
04747043	10	9	7	4	3	Multiprocessor cache coherence system
04785398	10	9	7	4	3	Virtual cache system using page level number generating CAM to access other memories for processing
04814981	10	9	7	4	3	Cache invalidate protocol for digital data processing system
04853846	10	9	7	4	3	Bus expander with logic for virtualizing single cache control into dual channels with separate directories
04992976	10	9	7	4	3	Method of allocating board slot numbers with altering software
04005389	30	9	7	4	3	Arrangement for reducing the access time in a storage system
04051461	30	9	7	4	3	Management table apparatus in memory hierarchy system

Patent #	50	40	30	20	10	Title
04157586	30	9	7	4	3	Technique for performing partial stores in store-thru memory configuration
04197580	30	9	7	4	3	Data processing system including a cache memory
04296465	30	9	7	4	3	Data mover
04339804	30	9	7	4	3	Memory system wherein individual bits may be updated
04400774	30	9	7	4	3	Cache addressing arrangement in a computer system
04433388	30	9	7	4	3	Longitudinal parity
04458310	30	9	7	4	3	Cache memory using a lowest priority replacement circuit
04493026	30	9	7	4	3	Set associative sector cache
04493033	30	9	7	4	3	Dual port cache with interleaved read accesses during alternate half-cycles and simultaneous writing
04680702	30	9	7	4	3	Merge control apparatus for a store into cache of a data processing system
04797813	30	9	7	4	3	Cache memory control apparatus
04827401	30	9	7	4	3	Method and apparatus for synchronizing clocks prior to the execution of a flush operation
04853891	30	9	7	4	3	Memory-programmable controller
04912630	30	9	7	4	3	Cache address comparator with sram having burst addressing control
04953079	30	9	7	4	3	Cache memory address modifier for dynamic alteration of cache block fetch sequence
03916384	26	22	17	11	3	Communication switching system computer memory control arrangement
04017840	26	22	17	11	3	Method and apparatus for protecting memory storage location accesses
04027291	26	22	17	11	3	Access control unit
04056844	26	22	17	11	3	Memory control system using plural buffer address arrays
04165531	26	22	17	11	3	Data generator for disc file addresses
04166289	26	22	17	11	3	Storage controller for a digital signal processing system
04214304	26	22	17	11	3	Multiprogrammed data processing system with improved interlock control
04371924	26	22	17	11	3	Computer system apparatus for prefetching data requested by a peripheral device from memory
04394734	26	22	17	11	3	Programmable peripheral processing controller
04408275	26	22	17	11	3	Data processing system with data cross-block-detection feature
04424562	26	22	17	11	3	Data processing system having plural address arrays
04446516	26	22	17	11	3	Data compaction system with contiguous storage of non-redundant information and run length counts
04447887	26	22	17	11	3	Method of rewriting data in non-volatile memory, and system therefor
04476522	26	22	17	11	3	Programmable peripheral processing controller with mode-selectable address register sequencing
04511962	26	22	17	11	3	Memory control unit
04562532	26	22	17	11	3	Main storage configuration control system
04658349	26	22	17	11	3	Direct memory access control circuit and data processing system using said circuit
04677544	26	22	17	11	3	Device for controlling access to a computer memory
04710894	26	22	17	11	3	Access control system for storage having hardware area and software area
04733349	26	22	17	11	3	Method for recording and managing processing history information using a plurality of storage devices
04733352	26	22	17	11	3	Lock control for a shared storage in a data processing system
04740889	26	22	17	11	3	Cache disable for a data processor
04766535	26	22	17	11	3	High-performance multiple port memory
04773026	26	22	17	11	3	Picture display memory system
04780815	26	22	17	11	3	Memory control method and apparatus
04783737	26	22	17	11	3	PROM writer adapted to accept new writing algorithm
04791564	26	22	17	11	3	Random access memory file apparatus for personal computer with external memory file
04887234	26	22	17	11	3	Portable electronic device with plural memory areas
04901273	26	22	17	11	3	Electronic postage meter having a memory map decoder
04901276	26	22	17	11	3	Portable electronic apparatus having a function of checking for empty areas in which to write data
04914575	26	22	17	11	3	System for transferring data between an interleaved main memory and an I/O device at high speed
04916609	26	22	17	11	3	Data processing system for processing units having different throughputs
04937740	26	22	17	11	3	Real time software analyzing system for storing selective m-bit addresses based upon correspondingly
04949244	26	22	17	11	3	Storage system
04956804	26	22	17	11	3	Data processing system with memories access time counting and information processor wait signal
04967391	26	22	17	11	3	Data string retrieval apparatus for IC card
05007011	26	22	17	11	3	Data storage device including data address predicting function
04214303	29	24	17	11	3	Word oriented high speed buffer memory system connected to a system bus
04292470	29	24	17	11	3	Audio signal recognition computer
04357656	29	24	17	11	3	Method and apparatus for disabling and diagnosing cache memory storage locations
04377844	29	24	17	11	3	Address translator
04426682	29	24	17	11	3	Fast cache flush mechanism
04442488	29	24	17	11	3	Instruction cache memory system
04502110	29	24	17	11	3	Split-cache having equal size operand and instruction memories
04638431	29	24	17	11	3	Data processing system for vector processing having a cache invalidation control unit
04689741	29	24	17	11	3	Video system having a dual-port memory with inhibited random access during transfer cycles
04794523	29	24	17	11	3	Cache memory architecture for microcomputer speed-up board
04835733	29	24	17	11	3	Programmable access memory
04847758	29	24	17	11	3	Main memory access in a microprocessor system with a cache memory
04884196	29	24	17	11	3	System for providing data for an external circuit and related method
04884198	29	24	17	11	3	Single cycle processor/cache interface
04901228	29	24	17	11	3	Pipelined cache system using back up address registers for providing error recovery while continuing
04912626	29	24	17	11	3	Hit predictive cache memory
04937738	29	24	17	11	3	Data processing system which selectively bypasses a cache memory in fetching information based upon
04942521	29	24	17	11	3	Microprocessor with a cache memory in which validity flags for first and second data areas are
04954944	29	24	17	11	3	Cache control circuit in cache memory unit with unit for enabling to reduce a read access time for cache
04991088	29	24	17	11	3	Method for optimizing utilization of a cache memory
03990055	27	23	18	12	3	Control system for magnetic disc storage device
04115855	27	23	18	12	3	Buffer memory control device having priority control units for priority processing set blocks and unit
04200928	27	23	18	12	3	Method and apparatus for weighting the priority of access to variable length data blocks in a multiple-disk
04215400	27	23	18	12	3	Disk address controller
04298929	27	23	18	12	3	Integrated multilevel storage hierarchy for a data processing system with improved channel to memory
04323968	27	23	18	12	3	Multilevel storage system having unitary control of data transfers
04399503	27	23	18	12	3	Dynamic disk buffer control unit
04580212	27	23	18	12	3	Computer having correctable read only memory
04682305	27	23	18	12	3	Storage system
04733367	27	23	18	12	3	Swap control apparatus for hierarchical memory system
04751638	27	23	18	12	3	Buffer storage control system having buffer storage unit comparing operand (OP) side and instruction
04755933	27	23	18	12	3	Data Processor system having look-ahead control
04819203	27	23	18	12	3	Control system for interruption long data transfers between a disk unit or disk coche and main memory to

Patent #	50	40	30	20	10	Title
04984149	27	23	18	12	3	Memory access control apparatus
04115851	28	23	18	12	3	Memory access control system
04136386	28	23	18	12	3	Backing store access coordination in a multi-processor system
04169284	28	23	18	12	3	Cache control for concurrent access
04441152	28	23	18	12	3	Data processing system having ring-like connected multiprocessors relative to key storage
04494215	28	23	18	12	3	Disk system
04547848	28	23	18	12	3	Access control processing system in computer system
04589064	28	23	18	12	3	System for controlling key storage unit which controls access to main storage
04658356	28	23	18	12	3	Control system for updating a change bit
04675811	28	23	18	12	3	Multi-processor system with hierarchy buffer storages
04688172	28	23	18	12	3	Initialization apparatus for a data processing system with a plurality of input/output and storage controller
04760546	28	23	18	12	3	Tag control circuit for increasing throughput of main storage access
04775955	28	23	18	12	3	Cache coherence mechanism based on locking
04843543	28	23	18	12	3	Storage control method and apparatus
04916604	28	23	18	12	3	Cache storage apparatus
04954982	28	23	18	12	3	Method and circuit for checking storage protection by pre-checking an access request key
04122530	31	25	19	13	3	Data management method and system for random access electron beam memory
04141067	31	25	19	13	3	Multiprocessor system with cache memory
04161024	31	25	19	13	3	Private cache-to-CPU interface in a bus oriented data processing system
04195342	31	25	19	13	3	Multi-configurable cache store system
04439829	31	25	19	13	3	Data processing machine with improved cache memory management
04514808	31	25	19	13	3	Data transfer system for a data processing system provided with direct memory access units
04558410	31	25	19	13	3	Microcomputer system for high-speed location of blocks of characteristics
04707784	31	25	19	13	3	Prioritized secondary use of a cache with simultaneous access
04713755	31	25	19	13	3	Cache memory consistency control with explicit software instructions
04716545	31	25	19	13	3	Memory means with multiple word read and single word write
04724518	31	25	19	13	3	Odd/even storage in cache memory
04747070	31	25	19	13	3	Reconfigurable memory system
04816997	31	25	19	13	3	Bus master having selective burst deferral
04823259	31	25	19	13	3	High speed buffer store arrangement for quick wide transfer of data
04853848	31	25	19	13	3	Block access system using cache memory
04858111	31	25	19	13	3	Write-back cache system using concurrent address transfers to setup requested address in main
04905188	31	25	19	13	3	Functional cache memory chip architecture for improved cache access
04910656	31	25	19	13	3	Bus master having selective burst initiation
04912631	31	25	19	13	3	Burst mode cache with wrap-around fill
04912632	31	25	19	13	3	Memory control subsystem
04914573	31	25	19	13	3	Bus master which selectively attempts to fill complete entries in a cache line
04918645	31	25	19	13	3	Computer bus having page mode memory access
04939641	31	25	19	13	3	Multi-processor system with cache memories
04244033	32	26	20	13	3	Method and system for operating an associative memory
04437149	32	26	20	13	3	Cache memory architecture with decoding
04442487	32	26	20	13	3	Three level memory hierarchy using write and share flags
04445174	32	26	20	13	3	Multiprocessing system including a shared cache
04583165	32	26	20	13	3	Apparatus and method for controlling storage access in a multilevel storage system
04719568	32	26	20	13	3	Hierarchical memory system including separate cache memories for storing data and instructions
04755930	32	26	20	13	3	Hierarchical cache memory system and method
04774654	32	26	20	13	3	Apparatus and method for prefetching subblocks from a low speed memory to a high speed memory of a
04807110	32	26	20	13	3	Prefetching system for a cache having a second directory for sequentially accessed blocks
04843542	32	26	20	13	3	Virtual memory cache for use in multi-processing systems
04885680	32	26	20	13	3	Method and apparatus for efficiently handling temporarily cacheable data
04914582	32	26	20	13	3	Cache tag lookaside
04956768	32	26	20	13	3	Wideband server, in particular for transmitting music or images
04980823	32	26	20	13	3	Sequential prefetching with deconfirmation
03911403	11	10	8	5	4	Data storage and processing apparatus
04068301	11	10	8	5	4	Data storage device comprising search means
04502127	11	10	8	5	4	Test system memory architecture for passing parameters and testing dynamic components
04799149	11	10	8	5	4	Hybrid associative memory composed of a non-associative basic storage and an associative surface, as
04956805	11	10	8	5	4	Circuitry for character translate functions
04008460	13	10	8	5	4	Circuit for implementing a modified LRU replacement algorithm for a cache
04093987	13	10	8	5	4	Hardware control storage area protection method and means
04280176	13	10	8	5	4	Memory configuration, address interleaving, relocation and access control system
04293910	13	10	8	5	4	Reconfigurable key-in-storage means for protecting interleaved main storage
04802086	13	10	8	5	4	FINUFO cache replacement method and apparatus
04907189	13	10	8	5	4	Cache tag comparator with read mode and compare mode
03938097	12	11	9	5	4	Memory and buffer arrangement for digital computers
04495565	12	11	9	5	4	Computer memory address matcher and process
04530049	12	11	9	5	4	Stack cache with fixed size stack frames
04648030	12	11	9	5	4	Cache invalidation mechanism for multiprocessor systems
04654787	12	11	9	5	4	Apparatus for locating memory modules having different sizes within a memory space
04679167	12	11	9	5	4	Apparatus for locating a memory module within a memory space
04698750	12	11	9	5	4	Security for integrated circuit microcomputer with EEPROM
04736293	12	11	9	5	4	Interleaved set-associative memory
04829420	12	11	9	5	4	Process and circuit arrangement for addressing the memories of a plurality of data processing units in a
04839796	12	11	9	5	4	Static frame digital memory
04860192	12	11	9	5	4	Quadword boundary cache system
04899275	12	11	9	5	4	Cache-MMU system
04035779	14	12	10	6	4	Supervisor address key control system
04037214	14	12	10	6	4	Key register controlled accessing system
04038645	14	12	10	6	4	Non-translatable storage protection control system
04042911	14	12	10	6	4	Outer and asynchronous storage extension system
04050060	14	12	10	6	4	Equate operand address space control system
04149245	14	12	10	6	4	High speed store request processing control
04189770	14	12	10	6	4	Cache bypass control for operand fetches
04317168	14	12	10	6	4	Cache organization enabling concurrent line castout and line fetch transfers with main storage
04654778	14	12	10	6	4	Direct parallel path for storage accesses unloading common system path

Patent #	50	40	30	20	10	Title
04142234	15	13	10	6	4	Bias filter memory for filtering out unnecessary interrogations of cache directories in a multiprocessor
04332010	15	13	10	6	4	Cache synonym detection and handling mechanism
04394731	15	13	10	6	4	Cache storage line shareability control for a multiprocessor system
04399506	15	13	10	6	4	Store-in-cache processor means for clearing main storage
04410946	15	13	10	6	4	Cache extension to processor local storage
04463420	15	13	10	6	4	Multiprocessor cache replacement under task control
04464712	15	13	10	6	4	Second level cache replacement method and apparatus
04472790	15	13	10	6	4	Storage fetch protect override controls
04484267	15	13	10	6	4	Cache sharing control in a multiprocessor
04503497	15	13	10	6	4	System for independent cache-to-cache transfer
04513367	15	13	10	6	4	Cache locking controls in a multiprocessor
04797814	15	13	10	6	4	Variable address mode cache
03914747	16	14	11	7	5	Memory having non-fixed relationships between addresses and storage locations
03982231	16	14	11	7	5	Prefixing in a multiprocessing system
04464713	16	14	11	7	5	Method and apparatus for converting addresses of a backing store having addressable data storage
04679139	16	14	11	7	5	Method and system for determination of data record order based on keyfield values
04839799	16	14	11	7	5	Buffer control method for quickly determining whether a required data block is in the buffer
04962466	16	14	11	7	5	Electronic product information display system
04044338	24	21	16	7	5	Associative memory having separately associable zones
04047160	24	21	16	7	5	Storage arrangement having a device to determine free storage locations
04291370	24	21	16	7	5	Core memory interface for coupling a processor to a memory having a differing word length
04296475	24	21	16	7	5	Word-organized, content-addressable memory
04298959	24	21	16	7	5	Digital information transfer system (DITS) receiver
04315312	24	21	16	7	5	Cache memory having a variable data block size
04374411	24	21	16	7	5	Relocatable read only memory
04437170	24	21	16	7	5	Method and circuit arrangement for the acceptance and temporary storage of data signals in a switching
04550367	24	21	16	7	5	Data processing system having hierarchical memories
04607331	24	21	16	7	5	Method and apparatus for implementing an algorithm associated with stored information
04685057	24	21	16	7	5	Memory mapping system
04700294	24	21	16	7	5	Data storage system having means for compressing input data from sets of correlated parameters
04803616	24	21	16	7	5	Buffer memory
04807179	24	21	16	7	5	Method and device for recording analog parameters on a static digital memory
04819204	24	21	16	7	5	Method for controlling memory access on a chip card and apparatus for carrying out the method
04845612	24	21	16	7	5	Apparatus for accessing a memory which has dedicated areas for separately storing addresses and
04977498	24	21	16	7	5	Data processing system having a data memory interlock coherency scheme
04232365	25	21	16	7	5	Apparatus for determining the next address of a requested block in interlaced rotating memories
04489378	25	21	16	7	5	Automatic adjustment of the quantity of prefetch data in a disk cache operation
04490782	25	21	16	7	5	I/O Storage controller cache system with prefetch determined by requested record's position within data
04502115	25	21	16	7	5	Data processing unit of a microprogram control system for variable length data
04573141	25	21	16	7	5	Memory interface for communicating between two storage media having incompatible data formats
04615018	25	21	16	7	5	Method for writing data into a memory
04628450	25	21	16	7	5	Data processing system having a local memory which does not use a directory device with distributed
04747038	25	21	16	7	5	Disk controller memory address register
04774687	25	21	16	7	5	Advanced store-in system for a hierarchy memory device
04845664	25	21	16	7	5	On-chip bit reordering structure
04864533	25	21	16	7	5	Data transfer control unit permitting data access to memory prior to completion of data transfer
04881167	25	21	16	7	5	Data memory system
04994962	25	21	16	7	5	Variable length cache fill
03956736	21	19	15	10	5	Disc cartridge sector formatting arrangement and record addressing system
04368513	21	19	15	10	5	Partial roll mode transfer for cyclic bulk memory
04734850	21	19	15	10	5	Data process system including plural storage means each capable of concurrent and intermediate
04809161	21	19	15	10	5	Data storage device
04958314	21	19	15	10	5	Information recording/reproducing apparatus
04047243	22	19	15	10	5	Segment replacement mechanism for varying program window sizes in a data processing system having
04052704	22	19	15	10	5	Apparatus for reordering the sequence of data stored in a serial memory
04092732	22	19	15	10	5	System for recovering data stored in failed memory unit
04607346	22	19	15	10	5	Apparatus and method for placing data on a partitioned direct access storage device
04680703	22	19	15	10	5	Data processing system with reorganization of disk storage for improved paging
04780808	22	19	15	10	5	Control of cache buffer for memory subsystem
04792898	22	19	15	10	5	Method and apparatus for temporarily storing multiple data records
04807180	22	19	15	10	5	Multiple control system for disk storage and method for realizing same
04811278	22	19	15	10	5	Secondary storage facility employing serial communications between drive and controller
04835678	22	19	15	10	5	Cache memory circuit for processing a read request during transfer of a data block
04888727	22	19	15	10	5	Peripheral controller with paged data buffer management
04903198	22	19	15	10	5	Method for substituting replacement tracks for defective tracks in disc memory systems
04916605	22	19	15	10	5	Fast write operations
04920478	22	19	15	10	5	Cache system used in a magnetic disk controller adopting an LRU system
04951194	22	19	15	10	5	Method for reducing memory allocations and data copying operations during program calling sequences
04972316	22	19	15	10	5	Method of handling disk sector errors in DASD cache
04437155	23	20	15	10	5	Cache/disk subsystem with dual aging of cache entries
04523206	23	20	15	10	5	Cache/disk system with writeback regulation relative to use of cache memory
04523275	23	20	15	10	5	Cache/disk subsystem with floating entry
04530054	23	20	15	10	5	Processor-addressable timestamp for indicating oldest written-to cache entry not copied back to bulk
04530055	23	20	15	10	5	Hierarchical memory system with variable regulation and priority of writeback from cache memory to bulk
04636974	23	20	15	10	5	Method and apparatus for transferring data between operationally-juxtaposed memories and knowledge-
04638424	23	20	15	10	5	Managing data storage devices connected to a digital computer
04771375	23	20	15	10	5	Managing data storage devices connected to a digital computer
04811203	23	20	15	10	5	Hierarchial memory system with separate criteria for replacement and writeback without replacement
03924245	17	15	12	8	6	Stack mechanism for a data processor
04333160	17	15	12	8	6	Memory control system
04706188	17	15	12	8	6	Method and apparatus for reading samples of a time-dependent signal in a data processing system
04734855	17	15	12	8	6	Apparatus and method for fast and stable data storage
04757469	17	15	12	8	6	Method of addressing a random access memory as a delay line, and signal processing device including
04799186	17	15	12	8	6	Electronic circuit constituting an improved high-speed stable memory with memory zones protect from
04807120	17	15	12	8	6	Temporal garbage collector with indirection cells

Patent #	50	40	30	20	10	Title
04817035	17	15	12	8	6	Method of recording in a disk memory and disk memory system
04833604	17	15	12	8	6	Method for the relocation of linked control blocks
04903195	17	15	12	8	6	Method for controlling data transfer
04984191	17	15	12	8	6	Limited write non-volatile memory and a franking machine making use thereof
04991112	17	15	12	8	6	Graphics system with graphics controller and DRAM controller
04293926	20	18	14	8	6	Dynamic type semiconductor memory equipment
04473890	20	18	14	8	6	Method and device for storing stereochemical information about chemical compounds
04128882	18	16	13	9	6	Packet memory system with hierarchical structure
04130885	18	16	13	9	6	Packet memory system for processing many independent memory transactions concurrently
04484262	18	16	13	9	6	Shared memory computer method and apparatus
04590586	18	16	13	9	6	Forced clear of a memory time-out to a maintenance exerciser
04652993	18	16	13	9	6	Multiple output port memory storage module
04707781	18	16	13	9	6	Shared memory computer method and apparatus
04187538	19	17	13	9	6	Read request selection system for redundant storage
04218756	19	17	13	9	6	Control circuit for modifying contents of packet switch random access memory
04332009	19	17	13	9	6	Memory protection system
04348720	19	17	13	9	6	Microcomputer arranged for direct memory access
04355356	19	17	13	9	6	Process and data system using data qualifiers
04356548	19	17	13	9	6	Process and data system using data extensions
04403300	19	17	13	9	6	Method and system of operation of an addressable memory permitting the identification of particular
04426681	19	17	13	9	6	Process and device for managing the conflicts raised by multiple access to same cache memory of a
04622631	19	17	13	9	6	Data processing system having a data coherence solution
04928234	19	17	13	9	6	Data processor system and method
04975870	19	17	13	9	6	Apparatus for locking a portion of a computer memory
03967247	37	30	22	15	7	Storage interface unit
04058851	37	30	22	15	7	Conditional bypass of error correction for dual memory access time selection
04070706	37	30	22	15	7	Parallel requestor priority determination and requestor address matching in a cache memory system
04092713	37	30	22	15	7	Post-write address word correction in cache memory system
04112502	37	30	22	15	7	Conditional bypass of error correction for dual memory access time selection
04228503	37	30	22	15	7	Multiplexed directory for dedicated cache memory system
04292674	37	30	22	15	7	One word buffer memory system
04381541	37	30	22	15	7	Buffer memory referencing system for two data words
04621320	37	30	22	15	7	Multi-user read-ahead memory
04363095	41	30	22	15	7	Hit/miss logic for a cache memory
04378591	41	30	22	15	7	Memory management unit for developing multiple physical addresses in parallel for use in a cache
04392201	41	30	22	15	7	Diagnostic subsystem for a cache memory
04424561	41	30	22	15	7	Odd/even bank structure for a cache memory
04445172	41	30	22	15	7	Data steering logic for the output of a cache memory having an odd/even bank structure
04520439	42	34	25	15	7	Variable field partial write data merge
04600986	42	34	25	15	7	Pipelined split stack with high performance interleaved decode
04674032	42	34	25	15	7	High-performance pipelined stack with over-write protection
04697233	42	34	25	15	7	Partial duplication of pipelined stack with data integrity checking
04757440	42	34	25	15	7	Pipelined data stack with access through-checking
04949301	42	34	25	15	7	Improved pointer FIFO controller for converting a standard RAM into a simulated dual FIFO by controlling
RE031318	38	31	23	16	8	Automatic modular memory address allocation system
04025903	38	31	23	16	8	Automatic modular memory address allocation system
04028678	39	32	23	16	8	Memory patching circuit
04028679	39	32	23	16	8	Memory patching circuit with increased capability
04028683	39	32	23	16	8	Memory patching circuit with counter
04028684	39	32	23	16	8	Memory patching circuit with repatching capability
04954951	40	33	24	17	8	System and method for increasing memory performance
RE033328	47	37	28	19	9	Write protect control circuit for computer hard disc systems
04005386	47	37	28	19	9	Clearing system
04234934	47	37	28	19	9	Apparatus for scaling memory addresses
04247905	47	37	28	19	9	Memory clear system
04295205	47	37	28	19	9	Solid state mass memory system compatible with rotating disc memory equipment
04296467	47	37	28	19	9	Rotating chip selection technique and apparatus
04507730	47	37	28	19	9	Memory system with automatic memory configuration
04545010	47	37	28	19	9	Memory identification apparatus and method
04731738	47	37	28	19	9	Memory timing and control apparatus
04734851	47	37	28	19	9	Write protect control circuit for computer hard disc systems
04757470	47	37	28	19	9	Pattern generation for a graphics display
04831513	47	37	28	19	9	Memory initialization system
04319324	50	40	28	19	9	Double word fetch system
04323965	50	40	28	19	9	Sequential chip select decode apparatus and method
04361869	50	40	28	19	9	Multimode memory system using a multiword common bus for double word and single word transfer
04366538	50	40	28	19	9	Memory controller with queue control apparatus
04366539	50	40	28	19	9	Memory controller with burst mode capability
04370712	50	40	28	19	9	Memory controller with address independent burst mode capability
04376972	50	40	28	19	9	Sequential word aligned address apparatus
04432055	50	40	28	19	9	Sequential word aligned addressing apparatus
04451880	50	40	28	19	9	Memory controller with interleaved queuing apparatus
04739473	50	40	28	19	9	Computer memory apparatus
04761730	50	40	28	19	9	Computer memory apparatus
04075686	48	38	29	20	10	Input/output cache system including bypass capability
04084234	48	38	29	20	10	Cache write capacity
04156906	48	38	29	20	10	Buffer store including control apparatus which facilitates the concurrent processing of a plurality of
04208716	48	38	29	20	10	Cache arrangement for performing simultaneous read/write operations
04217640	48	38	29	20	10	Cache unit with transit block buffer apparatus
04225922	48	38	29	20	10	Command queue apparatus included within a cache unit for facilitating command sequencing
04245304	48	38	29	20	10	Cache arrangement utilizing a split cycle mode of operation
04268907	48	38	29	20	10	Cache unit bypass apparatus
04312036	48	38	29	20	10	Instruction buffer apparatus of a cache unit
04313158	48	38	29	20	10	Cache apparatus for enabling overlap of instruction fetch operations
04314331	48	38	29	20	10	Cache unit information replacement apparatus

Patent #	50	40	30	20	10	Title
04157587	49	39	30	20	10	High speed buffer memory system with word prefetch
04167782	49	39	30	20	10	Continuous updating of cache store
04195340	49	39	30	20	10	First in first out activity queue for a cache store
04195341	49	39	30	20	10	Initialization of cache store to assure valid data
04195343	49	39	30	20	10	Round robin replacement for a cache store
04464717	49	39	30	20	10	Multilevel cache system with graceful degradation capability
04695943	49	39	30	20	10	Multiprocessor shared pipeline cache memory with split cycle and concurrent utilization
04768148	49	39	30	20	10	Read in process memory apparatus
04785395	49	39	30	20	10	Multiprocessor coherent cache system including two level shared cache with separately allocated
04833601	49	39	30	20	10	Cache resiliency in processing a variety of address faults
04992930	49	39	30	20	10	Synchronous cache memory system incorporating tie-breaker apparatus for maintaining cache

BAYES 3 - 10 Clusters

131	bayes3_10.1
190	bayes3_10.2
157	bayes3_10.3
44	bayes3_10.4
66	bayes3_10.5
31	bayes3_10.6
20	bayes3_10.7
7	bayes3_10.8
23	bayes3_10.9
22	bayes3_10.10
691	total

BAYES 3 - 20 Clusters

65	bayes3_20.1
115	bayes3_20.2
75	bayes3_20.3
34	bayes3_20.4
23	bayes3_20.5
21	bayes3_20.6
36	bayes3_20.7
14	bayes3_20.8
17	bayes3_20.9
30	bayes3_20.10
57	bayes3_20.11
29	bayes3_20.12
37	bayes3_20.13
46	bayes3_20.14
20	bayes3_20.15
6	bayes3_20.16
1	bayes3_20.17
20	bayes3_20.18
23	bayes3_20.19
22	bayes3_20.20
691	total

BAYES 3 - 30 Clusters

65	bayes3_30.1
37	bayes3_30.2
56	bayes3_30.3
19	bayes3_30.4
46	bayes3_30.5
32	bayes3_30.6
34	bayes3_30.7
11	bayes3_30.8
12	bayes3_30.9
21	bayes3_30.10
6	bayes3_30.11
12	bayes3_30.12
17	bayes3_30.13
2	bayes3_30.14
30	bayes3_30.15
30	bayes3_30.16
57	bayes3_30.17
29	bayes3_30.18
23	bayes3_30.19
14	bayes3_30.20
46	bayes3_30.21
14	bayes3_30.22
6	bayes3_30.23
1	bayes3_30.24
6	bayes3_30.25
15	bayes3_30.26
5	bayes3_30.27
23	bayes3_30.28
11	bayes3_30.29
11	bayes3_30.30
691	total

BAYES 3 - 40 Clusters

50	bayes3_40.1
37	bayes3_40.2
29	bayes3_40.3
19	bayes3_40.4
27	bayes3_40.5
32	bayes3_40.6
14	bayes3_40.7
32	bayes3_40.8
34	bayes3_40.9
11	bayes3_40.10
12	bayes3_40.11
9	bayes3_40.12
12	bayes3_40.13
6	bayes3_40.14
12	bayes3_40.15
6	bayes3_40.16
11	bayes3_40.17
2	bayes3_40.18
21	bayes3_40.19
9	bayes3_40.20
30	bayes3_40.21
37	bayes3_40.22
29	bayes3_40.23
20	bayes3_40.24
23	bayes3_40.25
14	bayes3_40.26
32	bayes3_40.27
14	bayes3_40.28
15	bayes3_40.29
14	bayes3_40.30
2	bayes3_40.31
4	bayes3_40.32
1	bayes3_40.33
6	bayes3_40.34
15	bayes3_40.35
5	bayes3_40.36
12	bayes3_40.37
11	bayes3_40.38
11	bayes3_40.39
11	bayes3_40.40
691	total

BAYES 3 - 50 Clusters

19	bayes3_50.1
37	bayes3_50.2
29	bayes3_50.3
15	bayes3_50.4
4	bayes3_50.5
27	bayes3_50.6
32	bayes3_50.7
14	bayes3_50.8
32	bayes3_50.9
17	bayes3_50.10
5	bayes3_50.11
12	bayes3_50.12
6	bayes3_50.13
9	bayes3_50.14
12	bayes3_50.15
6	bayes3_50.16
12	bayes3_50.17
6	bayes3_50.18
11	bayes3_50.19
2	bayes3_50.20
5	bayes3_50.21
16	bayes3_50.22
9	bayes3_50.23
17	bayes3_50.24
13	bayes3_50.25
37	bayes3_50.26
14	bayes3_50.27
15	bayes3_50.28
20	bayes3_50.29
17	bayes3_50.30
23	bayes3_50.31
14	bayes3_50.32
32	bayes3_50.33
14	bayes3_50.34
15	bayes3_50.35
31	bayes3_50.36
9	bayes3_50.37
2	bayes3_50.38
4	bayes3_50.39
1	bayes3_50.40
5	bayes3_50.41
6	bayes3_50.42
13	bayes3_50.43
4	bayes3_50.44
1	bayes3_50.45
2	bayes3_50.46
12	bayes3_50.47
11	bayes3_50.48
11	bayes3_50.49
11	bayes3_50.50
691	total

Section 3. First half to predict second half.

This section contains two CatHitTables. The first CatHitTable is for a test that used the first half of the Bayes3 data to train a Bayesian classifier, Bayes3a-Trained, that was applied to the second half of the Bayes3 data, Bayes3b. Bayes3a-Trained applied to Bayes3b had overall accuracy of 2680/3601 = 74.4%. The second CatHitTable is for a test that used the first half of the USPTO dat to train a Bayesian classifier, USPTOa-Trained, that was applied to the second half of the USPTO data, USPTOb. USPTOa-Trained applied to USPTOb had overall accuracy of 1969/3599 = 54.7%.

Bayes3a-Trained

Bayes3b

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Recall
0-0	(216)	162		3	3	1		1		1			2	3	16	6		5		13	75
1-1	(127)		80	1	16	2	4	1	2	1	9				1	8		1		1	63
2-2	(134)	3	6	78	10			6						14	3	12		1		1	58
3-3	(308)	14	15		239	7		2			1			4	1	19		1	2	3	78
4-4	(126)		2		13	85	1	3	4		6		1		3	7				1	67
5-5	(69)	8	1	3			26	3	1	1	3		2	1	4	12	1	3			38
6-6	(159)	7	1		15	4	2	114	4		1				2	2	1	1	1	4	72
7-7	(137)	1	3	2	7	1	5	5	58	4	7		2	9	3	16		2	5	7	42
8-8	(136)		5	2			6		1	96	7		4			2		13			71
9-9	(120)	1	5	1	1	2	2	1		3	86	6	6	1	1	2		2			72
10-10	(51)			1			8			3	4	17	8		3	4		2		1	33
11-11	(186)	2	1	3	1	1	1				2		149	17	5	1		1		2	80
12-12	(345)	10	2	30	7	1		2	1	8	2	1	10	246	2	11		5	2	5	71
13-13	(365)	4		3	2	1			3	1	1		5	5	335	2				3	92
14-14	(294)	7	4	6	20	2	1	2	1		1		1		1	239	1	5	3		81
15-15	(211)	12	1			1	5	1	1	15	4	3				2	153	12	1		73
16-16	(420)	1		2			1			4		1	2			7	5	396		1	94
17-17	(45)				1		1	1	3							3			36		80
18-18	(152)	18		1	4		2	3		5	1		4	4	17	3	1	3	1	85	56
Totals:	3601	250	126	136	339	108	65	145	79	142	135	28	196	304	397	358	162	453	51	127	2680
Precision:		65	63	57	71	79	40	79	73	68	64	61	76	81	84	67	94	87	71	67	

USPTOa-Trained

USPTOb

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Recall
0-0	(217)	118	6	1	20	1	2		1	4	2		6	9	25	8	1	6	1	6	54
1-1	(115)		44		25	1	4	2	5	2	13			4	4	8		3			38
2-2	(122)	8	6	35	14		1	3	1		1		2	26	7	12		3		3	29
3-3	(319)	28	35	4	153	13	3	4	3	2	6			20	2	42		1		3	48
4-4	(105)	3	5		11	44	3	4	5		10	1		2	9	6	1			1	42
5-5	(75)	6		1	7	2	23	3	1	3	1		4	2	5	6	3	5		3	31
6-6	(117)	9	3		12	5		58	4		4			4	6	4	1		2	5	50
7-7	(171)	5	5	2	11	7	3	8	54	7	10	1	5	18	8	9	1	5	3	9	32
8-8	(88)		6				3		1	53	5		2	3				15			60
9-9	(92)	4	9	2	3	3		2	2	2	47		3	5	8			1		1	51
10-10	(43)	1			2		2		1	3	9	5	4	2	5	3		5		1	12
11-11	(169)	8		1	2			1		2	1	1	96	41	13	1		1		1	57
12-12	(547)	30	2	18	49	11	2	11	5	17	7	4	64	269	17	15	1	14	3	8	49
13-13	(303)	10	1	2	3	2	3	2		1	3		8	11	237	4		3	1	12	78
14-14	(275)	10	16	5	50	9	2	5	6	3	6	1	2	8	3	129	2	14	2	2	47
15-15	(221)	14	2		1	1	2	1	2	12	5	1				1	154	25			70
16-16	(473)	3	1	3	2	1	5		2	12	3		3	4	1	10	6	417			88
17-17	(23)		2		3			1	5					1		1			10		43
18-18	(124)	18	1	1	8	3	2	4	3	3	6		6	9	23	7		6	1	23	19
Totals:	3599	275	144	75	376	103	60	109	101	126	139	14	205	438	373	266	170	524	23	78	1969
Precision:		43	31	47	41	43	38	53	53	42	34	36	47	61	64	48	91	80	43	29	

FORM CD-15
(12-6-73)
PRESCR. BY
DAO 214-2

U.S. DEPARTMENT OF COMMERCE

TRANSMIT/ROUTE

DATE

NAME	BUILDING, ROOM OR REFERENCE NO.	TAKE ACTION BELOW	INITIALS AND DATE
Deputy Commissioner Douglas B. Comer	PK2 Rm 904	1	
Deputy Assistant Comm/IS Boyd Alexander	PK2 Rm 916	2	3-28-91
Assistant Comm. for Patents James Denny	PK2 Rm 917	2	3/28/91
Director Grp 230 Gerald Goldberg	CP4 - 11D37	2	3/28/91

ACTION ITEMS

1. APPROVAL/SIGNATURE
2. CLEARANCE/INITIALS
3. RECOMMENDATION OR COMMENT
4. RETURN WITH MORE DETAILS
5. INVESTIGATE AND REPORT
6. NOTE AND SEE ME
7. NOTE AND RETURN
8. NOTE AND FILE
9. YOUR INFORMATION
10. PER OUR CONVERSATION
11. AS REQUESTED
12. NECESSARY ACTION
13. CIRCULATE AMONG STAFF
14. ANSWER DIRECTLY
15. PREPARE REPLY FOR SIGNATURE OF: ____________

COMMENTS

☐ *Continued on reverse*

FROM *(Name)*	BUILDING, ROOM OR REF. NO.	CODE AND EXTENSION
William Lawson Bill		557-0400

☆ U.S. GOVERNMENT PRINTING OFFICE: 1983 -403-545/7684

USCOMM-DC 1281-P74

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
ASSISTANT SECRETARY AND COMMISSIONER
OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

29 MAR 1991

Glen Self, Ph.D., JD.
Vice President
Research and Development
Electronic Data Systems Corporation
7223 Forest Lane
Dallas, Texas 75230-2398

Dear Dr. Self:

With reference to your February 19, 1991, letter to Commissioner Manbeck, we would like to proceed to test the automated patent classification system developed by EDS. From your letter and discussions you have had with Patent and Trademark Office (PTO) officials, it is our understanding that:

1. The PTO will provide to EDS

   o Classification (category) titles, definitions and listings of patent numbers contained therein for the nineteen new primary classifications developed to encompass the technology contained in US Patent Classification System Class 364, subclasses:

      200 & 900 (computer-related technology)
      513 & 513.5 (artificial intelligence)
      518 - 523 (data presentation)

   o On magnetic tape, the full text, including bibliographic data, for all US patents which have issued in the above subclasses since 1975 (about 15,000 patents), and documentation describing the magnetic tape data layout.

2. Using this information and data, and its automated patent classification software, EDS will

   o Construct subcategories within the thirteen primary categories and identify the subject matter and patents contained in those subcategories.

   o Conduct at its Albuquerque lab a demonstration of its system and the results of processing the PTO data for up to six PTO officials.

3. The PTO will

   - Send to EDS' Albuquerque lab up to six PTO officials, including examiners and classifiers in the technology, to participate in the demonstration. Two officials, an examiner and a classifier, will be sent a day or two in advance of the others to assist EDS.

   - Review the results of the demonstration and provide to EDS an evaluation, either "on the spot" in Albuquerque, and/or subsequently.

4. If the evaluation is favorable and upon request, EDS will provide to the PTO the software code and available documentation for their automated patent classification system, with the understanding that the PTO would have, at no charge, the rights to further use and modification of the software for its internal purposes and for the development of systems to which the public could have access without restriction or the charge of software license or usage fees.

Please let me know if these understandings are satisfactory to you. Assuming they are, we will begin to prepare and assemble the data and materials required so the test can proceed promptly. In the meantime, questions concerning details and scheduling can be addressed to Bill Lawson. His address is Administrator for Documentation; Patent and Trademark Office; CM2-300; Washington, DC 20231, Phone: 703-557-0400 FAX: 703-557-0668.

I look forward to hearing from you soon and to a successful test.

Sincerely,

Douglas B. Comer
Deputy Commissioner

Douglas B. Comer
Deputy Assistant Secretary and
Deputy Commissioner of Patents and Trademarks

Prepared by: WSLawson
bcc: J. Denny, B. Alexander, G. Goldberg, W. Lawson

File:TEST1/WP. Classification data refered to in Comer to Self, 29 Mar 91. See also files xxxxx.TXT.

CLASS 395 - INFORMATION SYSTEM PROCESSING ORGANIZATION

Schedule of Mainline Subclasses

5112H	ARTIFICIAL INTELLIGENCE
5114W	DATA PRESENTATION/COMPUTER GRAPHICS (E.G., IMAGE, GRAPHICS, TEXT)
5005D	TRANSMISSION OF INFORMATION AMONG MULTIPLE COMPUTER SYSTEMS
5007C	BUFFERING FUNCTIONS
5022E	I/O PROCESSING
5008H	SYSTEM INTERCONNECTIONS
5021T	INSTRUCTION PROCESSING
5019D	STORAGE ADDRESS FORMATION
5020W	STORAGE ACCESSING AND CONTROL
5011Y	COMPATIBILITY, SIMULATION, OR EMULATION OF SYSTEM COMPONENTS
5012C	TIMING
5013H	RELIABILITY
5015W	DATABASE OR FILE MANAGEMENT SYSTEM
5016T	PROCESSING (TASK) MANAGEMENT
5017E	SYSTEM UTILITIES
5010D	ACCESS CONTROL PROCESSING
5115T	POWER CONTROL
5116E	INTERNAL CONTROL
5000R	PROCESSING ARCHITECTURE

Definitions

Note: The following definitions have been edited to delete Search notes and other notes or parts of notes containing references to other classes or subclasses.

Class Definition

I. GENERAL STATEMENT OF THE CLASS SUBJECT MATTER

(1) This is the generic class for digital processing systems* and corresponding methods for performing information* processing functions*; and methods for controlling operations of such processing systems*.

(2) This class also provides for artificial intelligence, i.e., systems or methods which have the capacity to perform one or more of the functions of recognition, speech signal processing or knowledge processing (i.e., propositional logic, reasoning, learning, self-improvement), complex operations of a manipulator (e.g., robot control), or inexact reasoning (e.g., fuzzy logic).

(3) This class also provides for methods or apparatus for processing data for (1) static presentation (hard copy) or (2) display, wherein the processing of data for display includes the creation or manipulation of graphic objects (e.g., artificial image) or text by a computer prior to use with or in a specific display system.

III. GLOSSARY

The following terms have been defined for purposes of classification in this class. In the class and subclass definitions of this class, terms used in a sense defined below are indicated by an asterisk (*).

Computer system:
A digital processing system* which includes both a memory component and an I/O component.

Control program:
A computer program designed to schedule and to supervise the execution of programs on a digital processing system*.

Database:
A collection of permanently or semi-permanently stored data

which is related by at least one factor. The data is normally arranged in files which can be retrieved or manipulated by a user via a software interface generally referred to as a database manager.

Digital processing system:
An arrangement of processing component(s)* in combination with either memory component(s)* or I/O component(s)*, or both, which cooperate to accomplish information* processing functions(s)*.

End effector:
A terminal on a robot arm that carries a hand, welding gun, painting nozzle, or other tool.

Information:
Intelligence used to control and provide the basis for data processing. It includes address, data, control and status.

Input/output (I/O) component:
Apparatus which carries out necessary operations for transferring information* between processing components* and/or memory components*, and external components.

Instruction:
Coded information* for causing a processing component* to perform one or more of its operations.

Memory component:
Apparatus which provides storage of information*.

Processing component:
Apparatus which performs control, arithmetic, and/or logical operations.

Processing function:
Operations which cause a predefined sequence of events to occur.

Robot:
A robot is a powered arm structure having or capable of having an end effector*.

Subclasse Definitions

5112H ARTIFICIAL INTELLIGENCE:
Subject matter under the class definition wherein the system or method has the capacity to perform one or more of the functions of recognition, speech signal processing, knowledge processing (i.e., propositional logic, reasoning, learning, self-improvement), complex operations of a manipulator (e.g., robot* control), or inexact reasoning (e.g., fuzzy logic).

5114W DATA PRESENTATION/COMPUTER GRAPHICS (E.G., IMAGE, GRAPHICS, TEXT):
Subject matter under the class definition comprising methods or apparatus for processing data for (1) static presentation (hard copy) or (2) display, wherein the processing of data for display includes the creation or manipulation of graphic objects (e.g., artificial images) or text by a computer prior to use with or in a specific display system.

(1) Note. Graphic objects of this subclass type are defined by their coordinates, shape, size, and attributes. Such graphic objects define a portion of a displayed image and may be a combination of computer generated objects and real life images.

(4) Note. The use of a memory system for processing in conjunction with a data presentation/computer graphics system (e.g., for manipulating the addressing or contents of image or text information* stored in a memory) is classified herein.

5005D TRANSMISSION OF INFORMATION AMONG MULTIPLE COMPUTER SYSTEMS:
Subject matter under the class definition comprising apparatus or processes for transmitting information* (e.g., address, data, control and status) between multiple computer systems* via a communication medium and for performing significant processing in which there is a change in the content of the data before or after the transmission.

7007C BUFFERING FUNCTIONS:
Subject matter under the class definition including means or steps, involving buffering structure or operations, which change the speed of information* exchange among any combination of processing component(s)*, memory

component(s)* and I/O component(s)*.

5022E I/O PROCESSING:

Subject matter under the class definition including means or steps for the control of input/output components* or functions.

(1) Note. An input/output function of this subclass type carries out a necessary operation for transferring information* between processing or memory components* and external components of a digital processing system*, e.g., I/O processors, I/O controllers, I/O adapters, etc..

(2) Note. This subclass includes details of data transfers performed by an I/O component* to or from external components such as disk drives, peripheral devices, etc, and which involves an I/O function. This subclass does not include access to or from external components for storage (e.i., reading or writing) functions.

(3) Note. This subclass includes arbitration for interrupts from or to external or peripheral components.

(4) Note. This subclass does not include systems with specific physical devices external to the processing being done, or portable or handheld devices detachably connected to a computer system, where only nominal I/O functions are claimed, e.g., housings, circuit boards, keyboards, etc..

(5) This subclass does not include systems dealing with speech recognition or synthesis.

5008H SYSTEM INTERCONNECTIONS:

Subject matter under the class definition comprising details of means or steps relating to interconnections or communication between component(s) connected to an interconnection medium within a <u>single</u> digital processing system* dealing with digital information* processing.

(1) Note. This subclass includes arbitration for access to the interconnection medium (e.g., bus arbitration).

(2) Note. Subject matter including interface between compatible components only, i.e., components that can communicate information* without any format changes or translation are classified in this subclass.

(3) Note. Subject matter including bus cycling arrangements, crossbar switches and bus extenders are classified in this subclass.

5021T INSTRUCTION PROCESSING:
Subject matter under the class definition including details of instruction* prefetching, fetching, buffering, decoding, pipelining, and execution.

(1) Note. This subclass provides, for example, for different types of instructions* including macroinstruction and microinstruction.

5019D STORAGE ADDRESS FORMATION:
Subject matter under the class definition including means or methods for forming or manipulating memory addresses.

(1) Note. This subclass provides, for example, for virtual memory addressing, address translation, translation lookaside buffers (TLBs), boundary checking, and page-mode addressing.

5020W STORAGE ACCESSING AND CONTROL:
Subject matter under the class definition including the control of access to memory components* and the accessing of the memory components*.

(1) Note. This subclass provides for subject matter wherein static or dynamic storage is combined with significant data processing elements or techniques for accessing or preparing addresses.

(2) Note. This subclass provides, for example, for direct memory access (DMA) and arbitration for access to memory, plural memory element structure, memory protection, memory refreshing and cache memory.

5011Y COMPATIBILITY, SIMULATION OR EMULATION OF SYSTEM COMPONENTS:
Subject matter under the class definition including details of means or steps by which any one of a combination of processing component(s)*, memory component(s)* and I/O component(s)* is made compatible with at least one other processing component*, or made to simulate or emulate another processing component*.

(1) Note. This subclass provides, for example, for porting

application program(s) from one specific digital processing system* to another specific processing system*.

(2) Note. This subclass provides, for example, for different types of interfaces between incompatible components or programs that require data format changes to ensure proper communication or exchange of information*.

(3) Note. The inventive concept or the overall combination of the claimed system should be directed to performing compatibility operations.

(4) Note. Mere nominal recitation of compatibility, simulation or emulation means or steps are insufficient for classification in this subclass. This subclass provides for <u>how</u> these effects are accomplished rather than merely for the effects themselves.

5012C TIMING:

Subject matter under the class definition relating to the control or regulation of any one or combination of processing components*, memory components* and I/O components* according to a periodic sequence of timing pulses.

(2) This subclass provides, for example, for subject matter relating to timing operations in digital processing system(s)* including asynchronous time control, synchronous time control, time delay, cycle control and cycle steal.

5013H RELIABILITY:

Subject matter under the class definition including details of means or steps for diagnostic testing or monitoring of a digital processing system* for reliability purposes.

(1) Note. This subclass provides for power failure fail-safe functions, faultdetection/recovery and tracing/monitoring for failure detection or anticipation of a failure.

5015W DATABASE OR FILE MANAGEMENT SYSTEM

Subject matter under the class definition relating to the addressing, retrieval or manipulation of information* contained within the database* of a digital processing system*.

(1) Note. This subclass provides, for example, for file maintenance operations including insertion, deletion, updating, merge, collate, and sort.

(2) Note. This subclass provides, for example, for information* locating and retrieval techniques from a file which satisfies certain selection criteria based upon content or subject matter, specified addresses, or table lookup/linking.

(3) Note. This subclass provides, for example, for database management techniques including allocating/deallocating memory space to files, creating or searching file directory, and memory reclamation /garbage collection.

(4) Note. This subclass provides, for example, for database structures including distributed systems, sequential files, inverted files, hierarchical/tree filing systems, and relational systems.

5016T Processing (Task) Management:
Subject matter under the class definition relating to the control of operations of a plurality of cooperating processing components* or plurality of processes.

(1) Note. This subclass provides for procedural functions for task creating, scheduling, synchronization, and termination which include concurrent or simultaneous task operations, dead lock situations, load balancing, resource allocation, multitasking/multiprogramming techniques or scheduling techniques.

5017E System Utilities:
Subject matter under the class definition relating to the functions performed by an operating system (i.e., software for controlling computer operation).

(1) Note. This subclass provides for software that controls the execution of a computer program including, for example, assembler, disassembler, compiler, translator, editor and kernel/BIOS (Basic Input/Output System) software.

(2) Note. This subclass provides for computer system operation including, for example, booting/initialization, restarting/resetting, instructions minimization/optimization and reconfiguration.

(3) This subclass provides for operating system structure including distributed systems.

(4) Note. For purposes of classification in this subclass, interrupt processing is not considered a system utility.

5010D ACCESS CONTROL PROCESSING:
Subject matter under the class definition including steps or means for regulating availability of a digital processing system* resource.

(1) Note. This subclass provides, for example, for arbitration, interrupt handling, polling and lock/inhibiting schemes.

5115T POWER CONTROL
Subject matter under the class definition including details of steps or means for modifying the amount of power used by a computer system or the system response to available power.

(1) Note. This subclass includes power reduction, powering-up systems, powering-down systems, etc..

5116E INTERNAL CONTROL
Subject matter under the class definition including details of steps or means for controlling the interaction of elements within a processing component*.

(1) Note. This subclass includes control of operations within, for example, a CPU or ALU, i.e., between internal registers of the CPU and ALU.

5000R PROCESSING ARCHITECTURE:
Subject matter under the class definition which includes digital processing system(s)* having any combination or interconnection of processing component(s)*, memory component(s)* or I/O component(s)* not provided for above.

(1) Note. This subclass provides, for example, for MIMD, vector and array processors and single-chip microprocessors.

File: TEST2.WP

Proposed "Second" Test of the EDS System

1. The object of the proposed "second" test is to determine, in a quick and convenient way, the precision and recall ratios of the EDS system used as a classification tool. This can be accomplished using the results of work we've already done on our reclassification of Class 364.

2. To perform this test we would provide EDS with

   (A) the titles and definitions of the 5907C+ subclass array (see Attachment) and

   (B) a listing of the patents classified in the array (the xxxxxPLS.TXT files on the disk).

3. Using our definitions and titles as a source of words for identifying relevant patents, EDS would have its system classify these patents into our subclasses.

   NB: It will be necessary for EDS to take into account the hierarchical arrangement of our subclasses. One way this can be done is by screening all the patents for the first indented subclass (5908H), then screening the remainder for the second indented subclass (5979H) and so on through all the indented subclasses, assigning the ultimately remaining patents to the parent subclass (5907C). This sequential method could also be made a one-step process if, for instance, Boolean operators can be used in conjunction with the word list to assign patents identified for more than one subclass to the most superior indented subclass.

4. When all the patents have been assigned, lists of the patents in each subclass can be compared (by hand or by machine) with lists we will provide of the patents we assigned to each subclass during the reclassification project. The results can then be tabulated as precision and recall ratios.

Attachment

Subclass Array

5907C	.	Presentation processing
5908H	. .	Three-dimension
5979H	. .	Adjusting resolution or level of detail
5916T	. .	Surface detail/characteristic
5923N	. .	Object processing
5984H	. .	Generating graphs
5967W	. .	Generating shapes
5929W	. .	Text
5978C	. .	Animation
5940T	. .	Plural simultaneous presentations
5941E	. .	Operator interface (interactive)

Subclass Definitions

Note that the following subclass definition (for subclass 5114W), though not a part of the subclass array, is incorporated by reference in each of the definitions of the array.

5114W DATA PRESENTATION/COMPUTER GRAPHICS (E.G., IMAGE, GRAPHICS, TEXT):

Subject matter under the class definition comprising methods or apparatus for processing data for (1) static presentation (hard copy) or (2) display, wherein the processing of data for display includes the creation or manipulation of graphic objects (e.g., artificial images) or text by a computer prior to use with or in a specific display system.

(1) Note. Graphic objects of this subclass type are defined by their coordinates, shape, size, and attributes. Such graphic objects define a portion of a displayed image and may be a combination of computer generated objects and real life images.

5907C Presentation Processing:
Subject matter under subclass 5114W wherein data is displayed to a user for visual viewing (e.g., control of the form of the data on a CRT, monitor, screen, display device, or any generic visual output device).

(1) Note. Subject matter of this subclass type may include means or steps for interaction between a user and the display.

5908H Three-Dimension:
Subject matter under subclass 5114W wherein objects are pictured to show them as they appear to the eye with reference to relative distance or depth from an imaginary viewpoint.

5979H Adjusting Resolution or Level of Detail:
Subject matter under subclass 5114W wherein a change is made in the resolution of a graphic object.

(1) Note. In three dimensional systems resolution is used to account for distance or atmospheric conditions.

5916T Surface Detail/Characteristic:
Subject matter under subclass 5114W wherein the information is processed which relates to the attributes of an object or portion of an image.

5923N Object Processing:
Subject matter under subclass 5114W wherein the data representing a graphic object is manipulated.

(1) Note. This subclass provides for generating and processing data representing a graphic object.

5984H Generating Graphs:
Subject matter under subclass 5114W wherein a set of display points define a graph showing a relationship between two or more variables.

5967W Generating Shapes:
Subject matter under subclass 5114W wherein a set of display points define a two dimensional graphic object.

5929W Text:
Subject matter under subclass 5114W wherein the data to be presented is processed as a character stream.

5978C Animation:
Subject matter under subclass 5114W wherein a sequence of individual display images are generated for presentation as a sequence to simulate motion (e.g, cartoons).

5940T Plural Simultaneous Presentations:
Subject matter under subclass 5114W wherein the data is processed for multiple viewing outputs.

5941E Operator Interface (Interactive):
Subject matter under subclass 5114W wherein the user's inputs to a computer system are used to control the presentation of display data.

01/01/80 01:27 Directory A:*.*
Document Size: 0 Free Disk Space: 224256

. <CURRENT>		<DIR>			.. <PARENT>		<DIR>		
5000R	.TXT	5005	05/03/91	11:30	5005D	.TXT	2413	05/03/91	11:21
5007C	.TXT	3268	05/03/91	11:22	5008H	.TXT	7228	05/03/91	11:23
5010D	.TXT	2575	05/03/91	11:28	5011Y	.TXT	1477	05/03/91	11:25
5012C	.TXT	3421	05/03/91	11:26	5013H	.TXT	4213	05/03/91	11:26
5015W	.TXT	1900	05/03/91	11:27	5016T	.TXT	2035	05/03/91	11:27
5017E	.TXT	838	05/03/91	11:28	5019D	.TXT	3952	05/03/91	11:24
5020W	.TXT	15022	05/03/91	11:25	5021T	.TXT	7075	05/03/91	11:24
5022E	.TXT	7345	05/03/91	11:22	5112HPLS	.TXT	5005	05/03/91	11:12
5114WPLS	.TXT	10666	05/03/91	11:20	5115T	.TXT	487	05/03/91	11:28
5116E	.TXT	4375	05/03/91	11:29	5879C	.TXT	64	05/07/91	13:57
5907C	.TXT	136	05/07/91	13:48	5907CPLS	.TXT	5563	05/09/91	07:56
5908HPLS	.TXT	901	05/07/91	13:50	5916TPLS	.TXT	541	05/07/91	13:52
5923NPLS	.TXT	757	05/07/91	13:53	5929WPLS	.TXT	1414	05/07/91	13:57
5940TPLS	.TXT	154	05/07/91	13:58	5941EPLS	.TXT	712	05/07/91	13:59
5967WPLS	.TXT	613	05/07/91	13:55	5979H	.TXT	100	05/07/91	13:51
5984H	.TXT	181	05/07/91	13:53	TEST1	.WP	16020	01/01/80	01:19
TEST2	.WP	5740	01/01/80	01:27					

1 Retrieve; 2 Delete; 3 Rename; 4 Print; 5 Text In;
6 Look; 7 Change Directory; 8 Copy; 9 Word Search; 0 Exit: 6

Disk to Gallagher 7.30.91

More Communication 4704 Kunin

SAS Institute Inc.

SAS Campus Drive □ Cary, NC 27513
Phone 919.677.8000 □ Fax 919.677.4444
www.sas.com

February 29, 2000

Commissioner Q. Todd Dickinson
Assistant Secretary of Commerce and
Commissioner of Patents and Trademarks
United States Patent and Trademark Office
Office of the Commissioner
Washington, D.C. 20231

Re: Research Triangle Follow-up

Dear Commissioner Dickinson:

As a Board member of the Triangle Intellectual Property Law Association, I want to personally thank you for coming and speaking to our Research Triangle luncheon meeting on February 3, 2000. I commend you for your efforts in creating an outreach program with the business community, exemplified by the time you spent during your visit with several of the executives from various Triangle companies, particularly with Dr. Jim Goodnight and myself.

During the course of the outreach visit, Dr. Goodnight mentioned that SAS Institute published a manual in 1990 with technological references relevant to the Dickens Patent 5,806,063 ('063 patent). I am enclosing that material as promised.

Again, thank you for your willingness to speak to us in the Triangle and I hope to have the opportunity to visit with you and members of your staff in Washington, D.C. As an additional comment, Nicholas Flagler of your staff was extremely helpful. Thank you for putting him in charge of helping with this program.

Sincerely yours,

Mary U. Musacchia
Counsel to the President

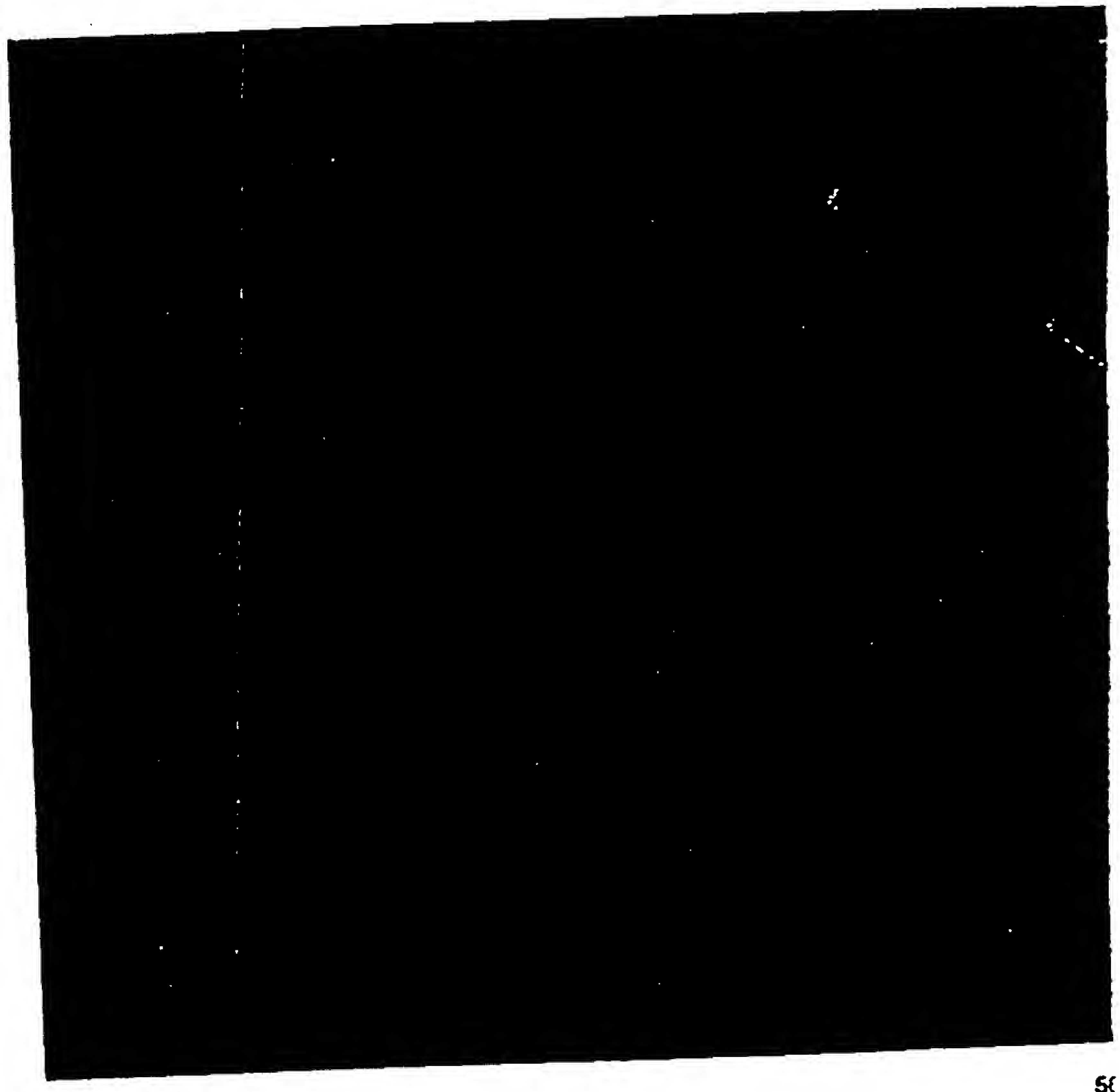
2000 MAR 14 PM 3:35
U.S. PATENT AND TRADEMARK OFFICE
ASSISTANT SECRETARY AND COMMISSIONER

MUM/LJK

Attachment

SAS® Language

Reference
Version 6
First Edition



SAS Institute Inc.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Language: Reference, Version 6, First Edition* Cary, NC: SAS Institute Inc., 1990. 1042 pp.

SAS® Language: Reference, Version 6, First Edition

Copyright © 1990 by SAS Institute Inc., Cary, NC, USA.

ISBN 1-55544-381-8

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

Restricted Rights Legend. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, March 1990
2nd printing, June 1992
3rd printing, June 1993
4th printing, July 1994
5th printing, September 1995

Note that text corrections may have been made at each printing.

The SAS® System is an integrated system of software providing complete control over data access, management, analysis, and presentation. Base SAS software is the foundation of the SAS System. Products within the SAS System include SAS/ACCESS®, SAS/AF®, SAS/ASSIST®, SAS/CALC®, SAS/CONNECT®, SAS/CPE®, SAS/DMI®, SAS/EIS®, SAS/ENGLISH®, SAS/ETS®, SAS/FSP®, SAS/GRAPH®, SAS/IMAGE®, SAS/IML®, SAS/IMS-DL/I®, SAS/INSIGHT®, SAS/LAB®, SAS/NVISION®, SAS/OR®, SAS/PH-Clinical®, SAS/QC®, SAS/REPLAY-CICS®, SAS/SESSION®, SAS/SHARE®, SAS/SPECTRAVIEW®, SAS/STAT®, SAS/TOOLKIT®, SAS/TRADER®, SAS/TUTOR®, SAS/DB2™, SAS/GEO™, SAS/GIS™, SAS/PH-Kinetics™, SAS/SHARE*NET™, and SAS/SQL-DS™ software. Other SAS Institute products are SYSTEM 2000® Data Management Software, with basic SYSTEM 2000, CREATE™, Multi-User™, QueX™, Screen Writer™, and CICS interface software; InfoTap® software; NeoVisuals® software; JMP®, JMP IN®, JMP Serve®, and JMP *Design®* software; SAS/RTERM® software; and the SAS/C® Compiler and the SAS/CX® Compiler; VisualSpace™ software; and Emulus® software. MultiVendor Architecture™ and MVA™ are trademarks of SAS Institute Inc. SAS Institute also offers SAS Consulting®, SAS Video Productions®, Ambassador Select®, and On-Site Ambassador™ services. *Authorline®*, Books by Users™, The Encore Series™, *JMPer Cable®*, *Observations®*, *SAS Communications®*, *SAS Training®*, *SAS Views®*, the SASware Ballot®, and SelecText™ documentation are published by SAS Institute Inc. The SAS Video Productions logo and the Books by Users SAS Institute's Author Service logo are registered service marks and the Helplus logo and The Encore Series logo are trademarks of SAS Institute Inc. All trademarks above are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

The Institute is a private company devoted to the support and further development of its software and related services.

Other brand and product names are registered trademarks or trademarks of their respective companies.

DOC S18, Ver 1.66W, 012290

WORKTERM

Erases WORK files at the termination of a SAS session

Valid as part of: configuration file, OPTIONS statement, OPTIONS window, SAS invocation

Syntax

WORKTERM | NOWORKTERM

Description

The WORKTERM system option specifies whether SAS WORK files, such as data sets, are erased from the current SAS WORK data library at the termination of the SAS session. The aliases for this option are WRKTERM and NOWRKTERM.

You can use the following forms of the WORKTERM system option:

WORKTERM specifies to erase the WORK files.

NOWORKTERM specifies not to erase the WORK files.

Although NOWORKTERM prevents the WORK data sets from being deleted, it has no effect on initialization of the WORK library by the SAS System. The SAS System normally initializes the WORK library at the start of each session, which effectively destroys any pre-existing information.

Comparisons

Use the WORKINIT system option to control whether the WORK data library is cleared when the SAS System is invoked.

See Also

WORKINIT system option

YEARCUTOFF=

Specifies the first year of a 100-year span used by informats and functions

Valid as part of: configuration file, OPTIONS statement, OPTIONS window, SAS invocation

HELP YEARCUTOFF

Syntax

YEARCUTOFF=*nnnn* | *nnnnn*

Description

The YEARCUTOFF= system option specifies the first year of a 100-year span used as the default by various DATE and DATETIME informats and functions.

You can use the following argument with the YEARCUTOFF= system option:

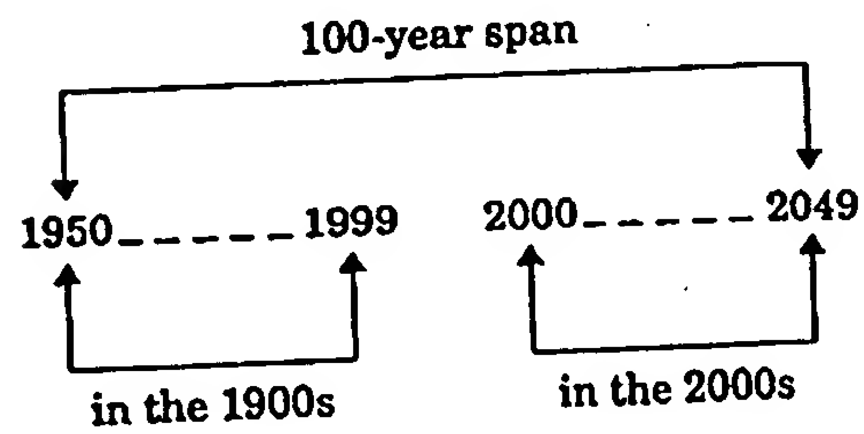
nnnn
nnnnn specifies the first year of the 100-year span. Valid values are from 1582 through 19900.

If the default value of *nnnn* (1900) is in effect, the 100-year span begins with 1900 and ends with 1999. Therefore, any informat or function that uses a two-digit year value assumes a prefix of 19. For example, the value 92 refers to the year 1992.

Note that the value specified in the YEARCUTOFF= system option can result in years that occur in two centuries. For example, if you specify YEARCUTOFF=1950, any two-digit value between 50 and 99 inclusive refers to the first half of the 100-year span, which is in the 1900s. Any two-digit value between 00 and 49 inclusive refers to the second half of

the 100-year span, which is in the 2000s. Figure 16.1 illustrates the relationship between the 100-year span and the two centuries if YEARCUTOFF=1950.

Figure 16.1 A 100-year Span with Values in Two Centuries



The YEARCUTOFF= system option applies to one- and two-digit years specified in the MMDDYY, YYMMDD, MONYY, DDMMYY, DATE, DATETIME, and YYQ informats and to one- and two-digit years specified in the DATEJUL, MDY, MONYY, and YYQ functions.

Host Information The syntax shown above applies to the OPTIONS statement. However, when you specify the YEARCUTOFF= system option on the command line or in a configuration file, the syntax is host specific and may include additional or alternate punctuation. For details, refer to the SAS documentation for your host system. ■

Computer processing of dates outside the twentieth century

by B. G. Ohms

A

This paper presents practical solutions to problems envisioned in extending computer processing of dates beyond the twentieth century. Many data processing managers are concerned with processing cross-century dates, and in doing so using existing systems, with a minimum of disruption to normal operations. The use of existing date formats can eliminate the need for massive system modifications. Methods of using existing date formats across century boundaries are explained. The use of a format termed the Lilian date format in honor of Luigi Lilio, the inventor of the Gregorian calendar, is introduced. The requirements for an effective date-processing algorithm are presented.

The Gregorian calendar serves us quite well in our day-to-day living. Due to discontinuities in various date divisions, however, it is not readily adaptable to computer programming. This fact becomes more apparent as we approach the new century. Few efficient, easy-to-use functions for manipulating dates have been produced. Also to be considered are the human requirements for ease of use, development, and maintenance. Other considerations include storage costs, efficiency, and adaptability across many different applications and environments.

Some early date-conversion programs were acts of expediency rather than planning, created to solve specific problems rather than for general programming use. Different functions were created at different times. Naming conventions, invocation formats, and implementation methods have often been inconsistent. Programs that provided a day-of-week function were rare. Also rare were programs for deriving a new date by adding or subtracting from another date in a different year. The functions that were available were normally not part of an integrated system for providing compatibility with most of the common date formats. Since documentation was not always created or maintained, individuals were often unaware of what was available. Today, dating format standards are being discussed internationally. The date-processing method presented in this paper is expected to be compatible with any foreseeable international standard date format as well as with the several formats discussed in this paper.

Typically, dates are displayed and stored in the *Julian* and *Gregorian* formats. Concepts and formats of both Julian and Gregorian date formats are discussed later in this paper. Simply stated, the Julian date is the number of the year together with the serial number of the day of that year. The Gregorian date format consists of month, day of month, and year. Many calendars show Julian dates printed in small numerals.

A format termed the *Lilian date format* is presented here as the basis for making date conversions of the

© Copyright 1986 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

type mentioned earlier. The Lilian format, which may be stored in three bytes of memory, provides the storage capacity for dates well beyond the year 10 000. This format handles processing across century years and other aspects of date conversion not currently adaptable to computer programming. Practical date processing services must provide ease of use for the end user, developer, and maintainer. Such services must also eliminate date ambiguity, achieve excellent performance, and minimize storage.

The two positions traditionally used in both Julian and Gregorian date formats implicitly represent a year within a century. However, this system is inadequate for representing dates in more than one century. For example, it is ambiguous as to whether 03 represents the year 1903 or the year 2003. The Lilian date format avoids the ambiguity by using seven positions for the number of days from the beginning of the Gregorian calendar, October 15, 1582.

Origins of date complexity

Julian calendar. The Julian calendar was established in 46 B.C. by Julius Caesar.[1] It replaced a system of constant adjustments, more for political reasons than for correcting inaccuracies in timekeeping. After a bad start, with the first few leap years being calculated incorrectly, it served quite adequately for many centuries. The Julian calendar is not the same as the Julian date, which was previously defined. The Julian calendar was a time-measuring system, which we now discuss.

The scheme of the Julian calendar was quite simple. The calendar year consisted of 365-day years, with years evenly divisible by four having 366 days. The resulting average year of 365.25 days was slightly longer than the *tropical year*, which is based on the time marked by the passage of equinoxes. That slight difference resulted in a gradual drift of the calendar year. The resulting difficulty in properly setting the date of celebration of Easter caused great concern to the Roman Church. The date for Easter is related to the date for Passover, which is related to the vernal equinox. By the sixteenth century, the discrepancy approximated ten days, and the desire for calendar reform intensified. An artifact of this discrepancy manifests itself today in the differences in dates for Christmas and Easter between the Western Church and the Eastern Church. The latter continues to use the Julian calendar dates as they were at the time of the Gregorian calendar reform. That is, the Eastern Church recognizes the Gregorian calendar, but for certain feast days does not void the ten-day error that existed at the time of the calendar reform.

Gregorian calendar. For the calendar reform, Pope Gregory XIII selected[2] the plan of Aloysius Lilius

England and her colonies adopted the Gregorian calendar on Thursday, September 14, 1752.

(1510–1576?),[1] who is also known as Luigi Lilio.[3] This reform, known as the Gregorian calendar, was implemented on Friday, October 15, 1582.[2]

Although use of the Gregorian calendar spread rapidly among Roman Catholic countries, many centuries passed before it was generally used by non-Catholic countries. England and her colonies, including what is now the United States, adopted the calendar on Thursday, September 14, 1752.[3] Many other countries—notably China, Greece, and Russia—did not adopt it until after the beginning of the 20th century. In fact, Russia adopted the Gregorian calendar on two separate occasions, first in 1918, about the same time as many other countries, and again on June 27, 1940.[4] Changes made in Russia in 1929 to avoid the religious associations of the Gregorian calendar were unsuccessful, and it was reimplemented in 1940.

The reform that synchronized the calendar year with the tropical year required the removal of ten days from the calendar. As a result, Friday, October 15, 1582, immediately followed Thursday, October 4, 1582. (An alternate plan would have removed the ten excess days gradually by canceling leap days for 40 years.) As before, every fourth year is a leap year, and, to maintain synchronization, centurial years that are evenly divisible by 400 are also leap years.[2] Thus, for example, 1900 was a common year; the year 2000 will be a leap year, and the year 2100 will start a series of common centurial years again. The result is an average calendar year of 365.2425 days, which closely approximates the tropical year. Despite

this close approximation, by the 44th century,[5] the Gregorian calendar will again differ from the tropical year by one day. Because the tropical year consists of an irrational number of days, no calendar year with an integral number of days can exactly match the tropical year. Not only is there not an integral number of days in a tropical year, but also the length of the day is not constant. That is, the length of the tropical year is also changing gradually.

Discontinuities in the Gregorian calendar scheme cause most of the difficulties with date manipulation.

Algorithms for date processing

Centurian date format. Discontinuities in the Gregorian calendar scheme cause most, if not all, of the difficulties associated with date manipulation. Instances of discontinuities are the following:

- No calendar unit is evenly divisible by weeks.
- The number of days per month varies.
- The number of days per year varies.
- The number of days per century varies.

These facts must be accounted for in calculating the number of days between two dates, calculating a date based on the addition or subtraction of days from another date, and calculating the day of the week for a date. A *centurian date format* (DDDDD, representing the day within a century) works well by giving continuous values within a century. Also, other date formats and day of week are readily calculable from this value. The centurian format is limited to a century and results in discrepancies when a century boundary is crossed. Consideration must be given to century years when making a leap year calculation. The DDDDD format will not span more than 273 years (or 179 years in a two-byte binary format). Also, a standard point of origin for the starting day must be defined.

Lilian date format. The Lilian date format consists of seven positions for the number of days from the beginning of the Gregorian calendar. Day one is Friday, October 15, 1582, and the value is incremented by one for each subsequent day. Based on this format are services supporting 44 experimental functions (or more if the three DMY, MDY, and YMD experimental versions of the Gregorian format are counted) synthesized from eight mnemonics.

Function-naming convention. Functions are named to promote easy recall and to suggest the activity to be performed. In the algorithm and experimental PL/I program discussed in this paper, each function name is synthesized from two 3-letter mnemonic parts. These mnemonic parts are defined as follows:

- CLL: compact Lilian date
- DAY: day of the week
- GRG: Gregorian date
- JUL: Julian date
- LIL: Lilian date
- SGR: short Gregorian date
- SJL: short Julian date
- VAL: validate a date

The first part is a description of the *target*, and the second part is a description of the *source*. VAL (validation) and DAY (day of week) can appear only in the target position of the function.

Arguments presented to any of the conversion functions are always presented in the same order: new date (target); old date (source); range date (control), when required; and Gregorian format [specifier(s)], when required. Table 1 illustrates the format for conversion arguments. In all instances, a return code is set.

The format descriptions in Table 1 refer to the type of data—D for day, M for month, and Y for year, as well as the number of positions (e.g., YY for two year positions)—that the data occupy. The origin of the term "Julian date" for the YYDDD format is given in Reference 6. Nevertheless, dates represented in the Julian format conform to the Gregorian calendar and not to the Julian calendar. The Julian date for Thursday, November 14, 1985, is 85318 (short form) and 1985318 (extended form).

Algorithms. The process of conversion between a Lilian format date and a Julian format date is now described. The algorithms are simpler to calculate by choosing a virtual base date rather than the real one. After the calculations are made, any discrepancies are resolved. In the following algorithms, the num-

Table 1 Examples of conversion function arguments

Argument	Description
target=JULSJL (source, 1925)	Conversion to a Julian (YYYYDDD) date from a short Julian (YYDDD) date within the range 1925–2024
target=JULGRG (source, DMY)	Conversion to a Julian (YYYYDDD) date from a Gregorian (DDMMYYYY) date
target=LILGRG (source, YMD)	Conversion of a Lilian (packed format) date from a Gregorian (YYYYMMDD) date
target=CLLJUL (source)	Conversion of a compact Lilian (binary format) date from a Julian date
target=SGRGRG (source, YMD, MDY)	Conversion of a short Gregorian (YYMMDD) date from a Gregorian (MMDDYYYY) date
CALL VALJUL (source)	Validation of a Julian date
target=DAYSJL (source 1925)	Day of week for a short Julian date

bers in parentheses are results of calculations made on the previously given date, November 14, 1985.

Extended Julian to Lilian. Given an extended Julian date (YYYYDDD), compute the corresponding Lilian date. First, compute the number of days from virtual January 1, 1501, to the start of the year being converted (1985 318 Julian).

- Subtract 1501 from the Julian year (484).
- Multiply the difference by 365.25 (i.e., the average number of days per year) (176 781.00).
- Truncate the result. The leap day is kept only for full four years (176 781).

Then compute the number of Julian calendar (not Julian format) days from October 15, 1582, to the date being converted.

- Subtract 29 872, i.e., the number of days between virtual January 1, 1501, and October 15, 1582 (146 909).
- Add the Julian days (+318 = 147 227).

Next, make the Gregorian calendar adjustments to this value.

- Subtract 1501 from the Julian year (484).
- Divide the difference by 100. One leap year is usually skipped each century (4.84).
- Truncate the result to obtain the number of whole leap days. Partial leap days do not exist (4).
- Subtract the number of whole leap days from the number of Julian calendar days (147 223).

Because one out of every four centuries keeps all of its leap years, use the virtual year 1201, which is the beginning of the four-century cycle that contains the year 1582, to compute the number of leap years up to the target date.

- Subtract 1201 from the Julian year. The first four-hundred-year cycle began with virtual year 1201 and ended in the real 1600 (784).
- Divide the difference by 400 because one century leap year per 400 years is kept (1.96).
- Truncate the result because partial leap days do not exist (1).
- Add these leap days back into the adjusted Julian calendar days (147 224).

This final result is the number of Gregorian calendar days from the beginning of the Gregorian calendar (October 15, 1582) to the date being converted. This result is the sought-for Lilian date.

Lilian to extended Julian. The purpose of this example is to show the procedure for converting a Lilian date to an extended Julian date. First, create a virtual Lilian date, with a starting point of January 1, 1201. Convert this result into a Julian calendar (not Julian format) date. Then convert the Julian calendar date to a Julian format date. The procedure is as follows (147224 Lilian):

- Add 139 444 to the Lilian date. This is the number of days from virtual January 1, 1201 (the start of a 400-year cycle) to October 15, 1582. The result is a pseudo-Lilian date (286668).
- Divide the pseudo-Lilian date by 36 524.25, the average number of days per century (7.85).
- Truncate the result to obtain the number of century leap days (7).

- Add the number of century leap days to the pseudo-Lilian date (286675).
- Divide the number of century leap days by 4 (1.75).
- Truncate the result to obtain the number of four-century leap days (1).
- Subtract the number of four-century leap days from the pseudo-Lilian date, because they are already included (286674).

The result is the number of full Julian calendar days from January 1, 1201, to the date being converted. We now convert this to an extended Julian format date.

- Divide full Julian calendar days by 365.25. This usually gives one less than the year for the date being converted (784.87).
- If there is no remainder from the division, subtract one from the quotient; otherwise, truncate the quotient to get the pseudo-Julian prior year (784).
- Multiply the pseudo-Julian prior year by 365.25 to determine the number of annual Julian calendar days prior to the year for the date being converted (286356.00).
- Truncate the number of annual Julian calendar days to eliminate partial leap days (286356).
- Subtract the number of truncated annual Julian calendar days from the full number of Julian calendar days to obtain the number of current Julian calendar days in the year of the date being converted (318).
- Add 1201 to the pseudo-Julian prior year to obtain the real Julian year, i.e., 1200 for years prior to 1200 plus one for the current year (+784 = 1985).
- Multiply the real Julian year by 1000 to put it into its proper Julian format position (1985000).
- Add the current Julian calendar days to the result to complete the Julian format date from the Lilian format date (1985318).

Ease of conversion

Accommodating end users. End users usually enter two digits for the year in a date and understand the ambiguity that this represents. Therefore, even at the turn of the century, to avoid adverse user reaction, programs must continue to function with only two digits for year. The inference of the year 1997 from 97 and 2003 from 03 must continue. For the exceptional case where the correct meaning could be 1897 and 1903, entry of all four digits may be required.

The month-day-year and day-month-year formats are ambiguous. Therefore, it might be advisable to continue presenting the date in its conventional U.S. format with a parenthetical explanation of the format—such as (MM/DD/YY)—to avoid the ambiguity.

It is not necessary to change date formats in files, because it is possible to change the programs only.

However, it may be necessary to provide a conversion function that receives a definition of the implied century as a parameter. An excellent way to do this unambiguously is to specify a year as the desired starting point of a 100-year range. For example, if the starting year for the range is specified as 1925, the starting year for the range is specified as 1925, dates with year digits of 25 through 99 would be between 1925 and 1999, and dates with year digits of 00 through 24 would lie between 2000 and 2024.

Accommodating systems support. The conversion of isolated files to new date formats presents a rather trivial problem. In most cases, however, it is not possible to isolate the process. All programs that access the modified data must be changed simultaneously. In some large systems, literally thousands of programs may be involved. In these large systems, it may be prudent to avoid the cost and risk of massive changes in a short period of time.

It is not necessary to change date formats in files, because it is possible to change the programs only, so that the implied century in a date is recognized. Of course, in the vast majority of cases, that is exactly what does take place. Dates familiarly and implicitly exist within the 100-year range beginning with 1900 or 1901. Thus it is necessary merely to modify the programs so that the 100-year range starts at a later date. A beginning date set eighty years prior to the current systems date may be a reasonable convention. This is well within the range now in use.

The significant feature of this approach is that everything need not be modified at once. The modifications may be made over a period of years during

normal program maintenance. Of course, as systems are maintained or replaced, it would be practical to implement full information date formats. Where systems contain dates that span a range of more than 100 years, the century must already have been carried. In the rare event that this is not true, immediate conversion is unavoidable. Fortunately, in most applications, we can deal with centuries as exceptions rather than as a common problem.

Computational considerations

Storage. The two main considerations pertaining to storage are (1) the cost of storing large quantities of data; and (2) the computational cost of converting records within computer files to larger date-field sizes. When millions of dates are stored, as they are in most business systems, every additional byte required to save a single date multiplies to millions of additional bytes of storage. The programming necessary to accommodate larger date-field sizes in records further complicates date conversion.

Putting bounds on data-storage costs at reasonable levels and reducing conversion complications are both achievable. The allocation of additional space to record four positions for year, rather than the traditional two, is not the only possibility. Many systems currently store dates internally in packed Julian format, requiring three bytes of storage. In packed format, a Lilian date or an extended Julian date (YYYYDDD) both require four bytes of storage. However, if a binary format were to be adopted, either form of date could be stored in the three bytes used at present.

A more restrictive situation exists for systems in which dates are stored in two bytes instead of three. These systems use a binary format to record centurian or other forms of dates. In the near term for these systems, it may be necessary to continue storing dates in only two bytes. This cannot be accommodated with a Julian format. However, it is possible to store a range of dates by taking advantage of the continuous characteristic of a Lilian date.

Using the Lilian date system, any date in a selected range of 179 years may be stored as follows. Assume that the selected range is January 1, 1901, through December 31, 2079. Find the Lilian value of December 31, 1900. Subtract this value from the Lilian value of the date to be stored. Convert the result to a 16-bit (two-byte) binary value and store the result. Reverse the process to restore the two-byte date to Lilian format. Continuing to store dates in two bytes should be considered only where the programming cost of increasing record sizes in an existing system is prohibitive. The decreasing cost of storage makes the use of the full Lilian or Julian format a practical possibility when an application is being modified.

In the experiments on which this paper is based, the three-byte Lilian format for data storage has been found to be preferable. Internally, in computer use, the continuous nature of the Lilian date accommodates the types of processing that normally take place within a program. In addition, for all three storage sizes, a consistent format (i.e., the all-Lilian format or the quasi-Lilian format) is maintained.

The continuous nature of the Lilian date accommodates the types of processing that normally take place.

Flexibility. In our experiments, the IBM System 360/370 assembly language was used for coding the date functions. However, the method is easily adaptable to other programming languages such as COBOL, PL/I, and RPG. The experimental functions were also made re-entrant for flexibility within on-line environments.

Validity. Ideally, the validation of a date is necessary only at the point of its manual entry into a system. Experience teaches us that it is not unusual for an interfacing system to provide invalid dates. Therefore, all dates passed to conversion functions must be validated. When an invalid date is encountered by the experimental system, a null value is returned to the invoking program and a return code is set to indicate the nature of the invalid condition.

Efficiency. Date conversions are used heavily within certain applications. Because of the impact on a single system or an installation, efficiency must be inherent in date-conversion programs. Storage requirements for external display and internal calculations by computer programs are expected to mo-

tivate a high volume of conversions. Widely used programs in high-volume environments must perform well and not consume excessive amounts of storage. A date-conversion program that is good in all other ways will not be widely used if it is an operational bottleneck.

For the program discussed in this paper, written in System 360/370 assembly language and implemented as a set of PL/I functions, the experimental results indicated good efficiency. The longest execution paths, including PL/I prologue, validation, conversion, and PL/I epilogue, are a little more than one hundred machine-language instructions. Although the functions appear to the invoking PL/I program to be separate programs, they are all actually provided within a single program that requires less than 4K bytes of storage.

Experimental results indicated good efficiency.

Concluding remarks

Programmers require date-processing functions that effectively handle applications for both the present and the future. For the present, it is sufficient to validate source dates, to convert from one traditional date format to another, and to perform addition or subtraction operations involving dates. For the future, additional functions are required that support processing restricted only by the limitations of the Gregorian calendar. These functions must be fully compatible with existing date-format and record-size restrictions. Massive conversion efforts should not be required to process and store dates outside the twentieth century. Also, end users should be able to continue to use existing two-digit date formats when interacting with computer systems. In all cases the programs that provide these services must be reliable, efficient in the use of both processor and storage, and flexible in application.

Programs that embody all these qualities have been written and tested experimentally. The application as written requires less than 4K bytes of storage and has an average execution path of fewer than 100 machine language instructions. Re-entrancy and the possibility of multilanguage implementation indicate excellent flexibility. This approach presents a practical method of processing dates that is compatible with any dating format standard.

Acknowledgments

No significant work is done in isolation. Over the years, I have experienced the inspiration, assistance, and patience of many individuals. My colleagues Tom Gauthier and Jim Willard first sparked my interest in date processing several years ago. Recently Alex Chang, Barb Henderson, Jack Henriksen, Fred Lange, Bob Lord, Libby Ross, Karen Seabury, and Billy Shih have patiently served as a sounding board, made helpful suggestions, and have been active supporters. Sandy Mink provided valuable editorial support. Many unnamed others have been involved in my experiment, including every member of my family. Their support is also greatly appreciated.

Cited references

1. G. Moyer, "Luigi Lilio and the Gregorian reform of the calendar," *Sky and Telescope* 64, No. 11, 418–419 (November 1982).
2. J. J. Bond, *Handy Book of Rules and Tables for Verifying Dates Within the Christian Era*, Russell & Russell, a Division of Atheneum House, Inc., New York (1966).
3. *The New Encyclopedia Britannica*, Encyclopedia Britannica, Inc., Chicago (1980), p. 602.
4. F. Parise, Editor, *The Book of Calendars*, Facts on Life, Inc., New York (1982).
5. G. Moyer, "The Gregorian calendar," *Scientific American* 246, No. 5, 144–152 (May 1982).
6. M. A. Covington, "A calendar for the ages," *PC Tech Journal* 3, No. 12, 136–142 (December 1985). Joseph Justice Saliger (1540–1609) named the Julian date in honor of his uncle Julius.

General references

G. Moyer, "Astronomical scrapbook—Notes on the Gregorian calendar reform," *Sky and Telescope* 64, No. 12, 530–533 (December 1982).

International Organization for Standardization, "Writing of calendar dates in all-numeric form," Reference No. ISO 2014-1976(E) (April 1976).

International Organization for Standardization, "Information processing interchange—Representation of ordinal dates," Reference No. ISO 2711-1973(E) (January 1973).

Bruce G. Ohms *IBM Information Systems Group, 301 Merritt 7, Norwalk, Connecticut 06856.* Mr. Ohms joined IBM in 1967 and is currently a senior programmer/analyst working in the area of applications systems development. Prior to his current assignment, he has held a variety of positions in programming, systems design, analysis, and development center implementations. He received an A.A.S. degree in data processing from Belleville Area College, Belleville, Illinois, and he attended Yale University.

Reprint Order No. G321-5274.

Intelligent Report Maintenance Using Dialogue Manager

"Best Copy Available"

Gary Browe demonstrates a date utility that simplifies maintenance of FOCEXECs that contain specific date periods.

Notice This Material May Be Protected By Copyright Law (Title 17 U.S. Code)

by Gary Browe

Are your reports ready for the year 2000? Do they automatically change when the accounting period changes? Are they flexible and easily maintained? If not, then your FOCEXECs could benefit from the FOCUS date utility program discussed in this article.

Calculating the current accounting period is often necessary. For example, your customer needs a second quarter report, and it is currently the third quarter. This is a common situation in financial reporting, as reports must change to fulfill date requirements. To make accounting reports more automatic, create a FOCEXEC called DATES that contains the Dialogue Manager commands shown in Figure 1.

Figure 1 - DATES FOCEXEC

```
-* ***********************************************************
-* Author: Gary C. Browe . Ford Motor Co.        File: DATES
-* Date:  12/89                                  Release: 5.8.2
-* System:  Any
-* Master(s): none
-* Function: Date calculations for FOCUS files
-*
-* Remarks:   All &variable dates are in the form MMDDYYYY. using a
-*            4 digit year.  This was done to avoid additional
-*            program logic to calculate dates between the 20th and
-*            21st century.
-*
-* Abbreviations:  REF - reference    MTH - month     BEG - beginning
-*                 CUR - current      YR  - year      END - ending
-*                 ACT - accounting   PRE - previous
-* ***********************************************************
-*
-* ***********************************************************
-* The &variable REFDATE is set outside this FOCEXEC to change the
-* reference date to a date other than the current date.  This is done
-* to fake FOCUS into "thinking" that it is some date other than today.
-* This is useful to "recreate" accounting reports for previous accounting
-* periods or preview future reports without changing program logic.
-* The reference date is typed in MMDDYY format and must refer to a date
-* from 01/01/1950 to 12/31/2049 for this focexec to work correctly.
-* ***********************************************************
-*
-IF &REFDATE.EXIST EQ 0 GOTO SETTODAY ELSE GOTO SETOTHER;
-SETTODAY
```

(continued)

(Figure 1 - DATES FOCEXEC continued)

```
-SET &BASEDATE = &MDY;
-GOTO CONTINUE
-SETOTHER
-SET &BASEDATE = &REFDATE;
-CONTINUE
-*
-* ********************************************************************
-* Current date settings.
-* Note: If the year is between 50 and 99, the date is assumed to be in
-*       the 20th century (1900-1999) else the 21st century (2000-2099)
-* ********************************************************************
-*
-SET &HEADINGDT    = EDIT (&BASEDATE, '99/99/99');
-SET &MTH          = EDIT (&BASEDATE, '99$$$$');
-SET &CURMTH       = &MTH;
-SET &YR           = EDIT (&BASEDATE, '$$$$99');
-SET &YR           = IF &YR GE 50 AND &YR LE 99 THEN 19|&YR
-                      ELSE 20|&YR;
-SET &CURYR        = &YR;
-SET &BEGCURMTH    = &MTH | 01 | &CURYR;
-SET &BEGCURYR     = 0101 | &CURYR;
-SET &MIDCURYR     = 0630 | &CURYR;
-SET &ENDCURYR     = 1231 | &CURYR;
-SET &CURQTR       = IF &MTH EQ 1 OR 2 OR 3 THEN 1
-                        ELSE IF &MTH EQ 4 OR 5 OR 6 THEN 2
-                        ELSE IF &MTH EQ 7 OR 8 OR 9 THEN 3
-                        ELSE 4;
-SET &CURQTRNAME   = IF &CURQTR EQ 1 THEN 'FIRST'
-                      ELSE IF &CURQTR EQ 2 THEN 'SECOND'
-                      ELSE IF &CURQTR EQ 3 THEN 'THIRD'
-                      ELSE 'FOURTH';
-*
-* ********************************************************************
-* Accounting Period:  Determine the accounting period which is the
-*                     previous month or quarter.
-* ********************************************************************
-*
-SET &ACTQTR       = IF &CURQTR - 1 EQ 0 THEN 4 ELSE &CURQTR - 1;
-SET &PREQTR       = &ACTQTR;
-SET &ACTQTRNAME   = IF &ACTQTR EQ 1 THEN 'FIRST'
-                      ELSE IF &ACTQTR EQ 2 THEN 'SECOND'
-                      ELSE IF &ACTQTR EQ 3 THEN 'THIRD'
-                      ELSE 'FOURTH';
-SET &PREQTRNAME   = &ACTQTRNAME;
-SET &MTH          = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH          = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &ACTMTH       = &MTH;
-SET &PREMTH       = &ACTMTH;
-SET &YR           = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &ACTYR        = &YR;
-SET &BEGACTMTH    = &MTH | 01 | &ACTYR;
-SET &MIDACTMTH    = &MTH | 15 | &ACTYR;
-SET &BEGPREMTH    = &BEGACTMTH;
-SET &MIDPREMTH    = &MIDACTMTH;
-SET &BEGACTQTR    = IF &ACTQTR EQ 1 THEN 0101 | &ACTYR
-                      ELSE IF &ACTQTR EQ 2 THEN 0401 | &ACTYR
-                      ELSE IF &ACTQTR EQ 3 THEN 0701 | &ACTYR
-                      ELSE 0901 | &ACTYR;
-SET &BEGPREQTR    = &BEGACTQTR;
-SET &ENDACTQTR    = IF &ACTQTR EQ 1 THEN 0331 | &ACTYR
-                      ELSE IF &ACTQTR EQ 2 THEN 0630 | &ACTYR
-                      ELSE IF &ACTQTR EQ 3 THEN 0930 | &ACTYR
-                      ELSE 1231 | &ACTYR;
-SET &ENDPREQTR    = &ENDACTQTR;
```

(continued)

(Figure 1 - DATES FOCEXEC continued)

```
-SET &BEGACTYR      = 0101 | &ACTYR;
-SET &MIDACTYR      = 0630 | &ACTYR;
-SET &ENDACTYR      = 1231 | &ACTYR;
-SET &JAN           = 0101 | &ACTYR;
-SET &FEB           = 0201 | &ACTYR;
-SET &MAR           = 0301 | &ACTYR;
-SET &APR           = 0401 | &ACTYR;
-SET &MAY           = 0501 | &ACTYR;
-SET &JUN           = 0601 | &ACTYR;
-SET &JUL           = 0701 | &ACTYR;
-SET &AUG           = 0801 | &ACTYR;
-SET &SEP           = 0901 | &ACTYR;
-SET &OCT           = 1001 | &ACTYR;
-SET &NOV           = 1101 | &ACTYR;
-SET &DEC           = 1201 | &ACTYR;
-* ****************************************************************
-* Previous year: is the accounting year - 1;
-*
-* Note: If the current month is January, the accounting year will be
-*       the previous year AND the previous year will be the accounting
-*       year - 1.  This is done to allow previous year and accounting
-*       year comparisons in Year End reports.
-* ****************************************************************
-SET &PREYR         = &ACTYR - 1;
-SET &BEGPREYR      = 0101 | &PREYR;
-SET &MIDPREYR      = 0630 | &PREYR;
-SET &ENDPREYR      = 1231 | &PREYR;
-* ****************************************************************
-* Calculate two months ago: which is the current month - 2;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &TWOMTHAGO     = &MTH | 01 | &YR;
-* ****************************************************************
-* Calculate three months ago: which is the current month - 3;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &THREEMTHAGO   = &MTH | 01 | &YR;
-* ****************************************************************
-* Calculate four months ago: which is the current month - 4;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &FOURMTHAGO    = &MTH | 01 | &YR;
-* ****************************************************************
-* Calculate five months ago: which is the current month - 5;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &FIVEMTHAGO    = &MTH | 01 | &YR;
-* ****************************************************************
-* Calculate six months ago: which is the current month - 6;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &SIXMTHAGO     = &MTH | 01 | &YR;
```

These &variables, when resolved, will contain dates based on the system date or a reference date contained in the report request.

To understand how these &variables are resolved and what values they will contain after execution, execute the TESTDATE FOCEXEC shown in Figure 2 using the ECHO=ALL option (EX TESTDATE ECHO=ALL). The resolved &variables, shown in Figure 3, can then be used in screening statements, DEFINE or COMPUTE statements, column headings, or any other valid statements used in FOCUS.

Figure 2 - TESTDATE FOCEXEC

```
-* ******************************************************************
-* Author: Gary C. Brove . Ford Motor Co.     File: TESTDATE
-* Date:  11/89                               Release: 5.5.2
-* System:  Any
-* Master(s): none
-* Function:  Test focexec to type the resolved &variables:
-* ******************************************************************
-* &REFDATE must be typed in MMDDYY format. If the reference date is
-* omitted, the system date will be substituted.
-PROMPT &WHATDATE.[illegible] TYPE THE REFERENCE DATE IN MMDDYY FORMAT.
-SET &REFDATE = &WHATDATE;
-INCLUDE DATES[illegible]
-TYPE HEADINGDT       -  &HEADINGDT
-TYPE CURMTH          -  &CURMTH
-TYPE CURYR           -  &CURYR
-TYPE BEGCURMTH       -  &BEGCURMTH
-TYPE BEGCURYR        -  &BEGCURYR
-TYPE MIDCURYR        -  &MIDCURYR
-TYPE ENDCURYR        -  &ENDCURYR
-TYPE CURQTR          -  &CURQTR
-TYPE CURQTRNAME      -  &CURQTRNAME
-TYPE ACTQTR          -  &ACTQTR
-TYPE PREQTR          -  &PREQTR
-TYPE ACTQTRNAME      -  &ACTQTRNAME
-TYPE PREQTRNAME      -  &PREQTRNAME
-TYPE ACTMTH          -  &ACTMTH
-TYPE PREMTH          -  &PREMTH
-TYPE ACTYR           -  &ACTYR
-TYPE BEGACTMTH       -  &BEGACTMTH
-TYPE MIDACTMTH       -  &MIDACTMTH
-TYPE BEGPREMTH       -  &BEGPREMTH
-TYPE MIDPREMTH       -  &MIDPREMTH
-TYPE BEGACTQTR       -  &BEGACTQTR
-TYPE BEGPREQTR       -  &BEGPREQTR
-TYPE ENDACTQTR       -  &ENDACTQTR
-TYPE ENDPREQTR       -  &ENDPREQTR
-TYPE BEGACTYR        -  &BEGACTYR
-TYPE MIDACTYR        -  &MIDACTYR
-TYPE ENDACTYR        -  &ENDACTYR
-TYPE JAN             -  &JAN
-TYPE FEB             -  &FEB
-TYPE MAR             -  &MAR
-TYPE APR             -  &APR
-TYPE MAY             -  &MAY
-TYPE JUN             -  &JUN
-TYPE JUL             -  &JUL
-TYPE AUG             -  &AUG
-TYPE SEP             -  &SEP
-TYPE OCT             -  &OCT
-TYPE NOV             -  &NOV
-TYPE DEC             -  &DEC
```

(continued)

(Figure 2 - TESTDATE FOCEXEC continued)

```
-TYPE PREYR        -  &PREYR
-TYPE BEGPREYR     -  &BEGPREYR
-TYPE MIDPREYR     -  &MIDPREYR
-TYPE ENDPREYR     -  &ENDPREYR
-TYPE TWOMTHAGO    -  &TWOMTHAGO
-TYPE THREEMTHAGO  -  &THREEMTHAGO
-TYPE FOURMTHAGO   -  &FOURMTHAGO
-TYPE FIVEMTHAGO   -  &FIVEMTHAGO
-TYPE SIXMTHAGO    -  &SIXMTHAGO
-EXIT
```

Figure 3 - Resolved &variables from TESTDATE FOCEXEC

```
HEADINGDT     -  01/01/90
CURMTH        -  01
CURYR         -  1990
BEGCURMTH     -  01011990
BEGCURYR      -  01011990
MIDCURYR      -  06301990
ENDCURYR      -  12311990
CURQTR        -  1
CURQTRNAME    -  FIRST
ACTQTR        -  4
PREQTR        -  4
ACTQTRNAME    -  FOURTH
PREQTRNAME    -  FOURTH
ACTMTH        -  12
PREMTH        -  12
ACTYR         -  1989
BEGACTMTH     -  12011989
MIDACTMTH     -  12151989
BEGPREMTH     -  12011989
MIDPREMTH     -  12151989
BEGACTQTR     -  09011989
BEGPREQTR     -  09011989
ENDACTQTR     -  12311989
ENDPREQTR     -  12311989
BEGACTYR      -  01011989
MIDACTYR      -  06301989
ENDACTYR      -  12311989
JAN           -  01011989
FEB           -  02011989
MAR           -  03011988
APR           -  04011989
MAY           -  05011989
JUN           -  06011989
JUL           -  07011989
AUG           -  08011989
SEP           -  09011989
OCT           -  10011989
NOV           -  11011989
DEC           -  12011989
PREYR         -  1988
BEGPREYR      -  01011988
MIDPREYR      -  06301988
ENDPREYR      -  12311988
TWOMTHAGO     -  11011989
THREEMTHAGO   -  10011989
FOURMTHAGO    -  09011989
FIVEMTHAGO    -  08011989
SIXMTHAGO     -  07011989
```

The DATES procedure is called into the report request via a -INCLUDE statement. The resolved &variables are then available for use (see Figure 4). For more complex reports or special conditions, other Dialogue Manager &variables can be defined using the DATES procedure as a base program (see Figure 5).

When using this technique, bear in mind that a limited number of Dialogue Manager variables are available. This technique makes extensive use of these variables; before using it, ensure that your existing application is not already at or near the limit.

In summary, using this Dialogue Manager DATES procedure throughout your database management system can reduce or eliminate FOCEXEC maintenance during major date changes. It will also have the added benefit of making your FOCEXECs more uniform, thus making them easier to interpret and maintain. ✦

Figure 4 - Sample FOCEXEC

```
-*
-*************************************************************************
-* Author: Gary C. Browe , Ford Motor Co.      File: FIGURE4
-* Date:  12/89                                Release: 5.5.2
-* System:  Any
-* Master(s): EMPLOYEE
-* Function:  Sample example focexec using the DATES include file.
-* Remarks:  The output from this focexec was based on the EMPLOYEE
-*           database supplied with PC/FOCUS release 3.1
-*
-*************************************************************************
-SET &REFDATE = 080182 ;
-INCLUDE DATES
-*
-* ***********************************************************************
-* Define XPAY_DATE as a FOCUS date.
-* ***********************************************************************
-*
DEFINE FILE EMPLOYEE
XPAY_DATE/MDYY = PAY_DATE;
END
-*
-* ***********************************************************************
-* Produce the monthly report from the employee database.
-* ***********************************************************************
TABLE FILE EMPLOYEE
HEADING CENTER
"&HEADINGDT"
"MONTHLY REPORT OF SALARY INCREASE"
"FOR &ACTMTH / &ACTYR </1"
PRINT
 XPAY_DATE AS 'PAY,DATE'
 GROSS     AS 'GROSS,PAY'
BY LAST_NAME  AS 'LAST,NAME'
BY FIRST_NAME AS 'FIRST,NAME'
IF XPAY_DATE GE '&BEGACTMTH'
IF XPAY_DATE LT '&BEGCURMTH'
ON TABLE SUBFOOT
"<46 ________"
"TOTAL GROSS <42 <TOT.GROSS "
END
-EXIT
```

Output

```
                    08/01/82
        MONTHLY REPORT OF SALARY INCREASE
                  FOR 07 / 1982

LAST          FIRST       PAY                 GROSS
NAME          NAME        DATE                PAY

BLACKWOOD     ROSEMARIE   07/30/1982          $1,815.00
CROSS         BARBARA     07/30/1982          $2,255.00
IRVING        JOAN        07/30/1982          $2,238.50
JONES         DIANE       07/30/1982          $1,540.00
MCKNIGHT      ROGER       07/30/1982          $1,342.00
ROMANS        ANTHONY     07/30/1982          $1,760.00
SMITH         MARY        07/30/1982          $1,100.00
STEVENS       ALFRED      07/30/1982            $916.67

TOTAL GROSS                                  $12,967.17
```

Figure 5 - Sample FOCEXEC

```
-*
-*************************************************************************
-* Author: Gary C. Browe . Ford Motor Co.      File: FIGURES
-* Date:  11/89                                Release: 5.6.2
-* System:  Any
-* Master(s): EMPLOYEE
-* Function:  Example focexec using dates file with additional dates
-*            calculated.
-* Remarks:  The output from this focexec was based on the EMPLOYEE
-*           database supplied with PC/FOCUS release 3.1
-*
-*************************************************************************
-SET &REFDATE = 080183 ;
-INCLUDE DATES
-SET &2YRSAGO      = &PREYR - 1;
-SET &BEG2YRSAGO   = 0101 | &2YRSAGO;
-SET &END2YRSAGO   = 1231 | &2YRSAGO;
-*
-* ************************************************************************
-* Define XPAY_DATE as a FOCUS date
-* ************************************************************************
-*
DEFINE FILE EMPLOYEE
 XPAY_DATE/MDY = PAY_DATE;
 2YRSAGOPAY/D10.2M = IF XPAY_DATE GE '&BEG2YRSAGO'
                AND XPAY_DATE LE '&END2YRSAGO' THEN GROSS ELSE 0;
 PREYRPAY/D10.2M = IF XPAY_DATE GE '&BEGPREYR'
                AND XPAY_DATE LE '&ENDPREYR' THEN GROSS ELSE 0;
 ACTYRPAY/D10.2M = IF XPAY_DATE GE '&BEGACTYR'
                AND XPAY_DATE LE '&ENDACTYR' THEN GROSS ELSE 0;
END
TABLE FILE EMPLOYEE
HEADING CENTER
"&HEADINGDT
"EMPLOYEE PAY HISTORY REPORT
SUM
 2YRSAGOPAY AS '&2YRSAGO,GROSS,EARNINGS'
 PREYRPAY AS '&PREYR,GROSS,EARNINGS'
 ACTYRPAY AS '&ACTYR,GROSS,EARNINGS'
BY LAST_NAME AS 'LAST,NAME'
BY FIRST_NAME AS 'FIRST,NAME'
ON TABLE COLUMN-TOTAL
END
-EXIT
```

Output

08/01/83
EMPLOYEE PAY HISTORY REPORT

LAST NAME	FIRST NAME	1981 GROSS EARNINGS	1982 GROSS EARNINGS	1983 GROSS EARNINGS
BLACKWOOD	ROSEMARIE	$.00	$1,815.00	$.00
CROSS	BARBARA	$.00	$2,255.00	$.00
IRVING	JOAN	$.00	$2,238.50	$.00
JONES	DIANE	$.00	$1,540.00	$.00
MCKNIGHT	ROGER	$.00	$1,342.00	$.00
ROMANS	ANTHONY	$.00	$1,760.00	$.00
SMITH	MARY	$.00	$1,100.00	$.00
STEVENS	ALFRED	$.00	$916.67	$.00
TOTAL		$.00	$12,967.17	$.00

Gary Browe is a programmer analyst at Ford Motor Company, Parts and Service Division and president of the Detroit-Toledo FOCUS Users Group, TED McFUSE. His experience includes application design, end-user consulting, and giving FOCUS application presentations to FUSE groups. He has been using FOCUS since 1983.

Figure 5 - Sample FOCEXEC

```
-*
-*************************************************************************
-* Author: Gary C. Browe , Ford Motor Co.        File: FIGURES
                                                 Release: 5.5.2
-* Date:   11/89
-* System:  Any
-* Master(s): EMPLOYEE
-* Function:  Example focexec using dates file with additional dates
-*            calculated.
-* Remarks:  The output from this focexec was based on the EMPLOYEE
-*           database supplied with PC/FOCUS release 5.1
-*
-*************************************************************************
-SET &REFDATE = 080183 ;
-INCLUDE DATES
-SET &2YRSAGO      = &PREYR - 1;
-SET &BEG2YRSAGO   = 0101 | &2YRSAGO;
-SET &END2YRSAGO   = 1231 | &2YRSAGO;
-*
-*************************************************************************
-* Define XPAY_DATE as a FOCUS date.
-*************************************************************************
-*
DEFINE FILE EMPLOYEE [illegible]
XPAY_DATE/MDY = PAY_DATE;
2YRSAGOPAY/D10.2M = IF XPAY_DATE GE &BEG2YRSAGO
                AND XPAY_DATE LE &END2YRSAGO THEN GROSS ELSE 0;
PREYRPAY/D10.2M = IF XPAY_DATE GE &BEGPREYR
                AND XPAY_DATE LE &ENDPREYR THEN GROSS ELSE 0;
ACTYRPAY/D10.2M = IF XPAY_DATE GE &BEGACTYR
                AND XPAY_DATE LE &ENDACTYR THEN GROSS ELSE 0;
END
TABLE FILE EMPLOYEE
HEADING CENTER
"&HEADINGDT"
"EMPLOYEE PAY HISTORY REPORT </1"
SUM
 2YRSAGOPAY AS '&2YRSAGO,GROSS,EARNINGS'
 PREYRPAY AS '&PREYR,GROSS,EARNINGS'
 ACTYRPAY AS '&ACTYR,GROSS,EARNINGS'
BY LAST_NAME AS 'LAST,NAME'
BY FIRST_NAME AS 'FIRST,NAME'
ON TABLE COLUMN-TOTAL
END
[illegible]
```

Output:

```
                          08/01/83
                 EMPLOYEE PAY HISTORY REPORT
```

LAST NAME	FIRST NAME	1981 GROSS EARNINGS	1982 GROSS EARNINGS	1983 GROSS EARNINGS
BLACKWOOD	ROSEMARIE	$.00	$1,815.00	$.00
CROSS	BARBARA	$.00	$2,255.00	$.00
IRVING	JOAN	$.00	$2,238.50	$.00
JONES	DIANE	$.00	$1,540.00	$.00
MCKNIGHT	ROGER	$.00	$1,342.00	$.00
ROMANS	ANTHONY	$.00	$1,760.00	$.00
SMITH	MARY	$.00	$1,100.00	$.00
STEVENS	ALFRED	$.00	$916.67	$.00
TOTAL		$.00	$12,967.17	$.00

Gary Browe is a programmer analyst at Ford Motor Company, Parts and Service Division and president of the Detroit-Toledo FOCUS Users Group, TED McFUSE. His experience includes application design, end-user consulting, and giving FOCUS application presentations to FUSE groups. He has been using FOCUS since 1983.

MANAGER

Intelligent Report Maintenance Using Dialogue Manager

"Best Copy Available"

Gary Browe demonstrates a date utility that simplifies maintenance of FOCEXECs that contain specific date periods.

Notice This Material May Be Protected By Copyright Law (Title 17 U.S. Code)

by Gary Browe

Are your reports ready for the year 2000? Do they automatically change when the accounting period changes? Are they flexible and easily maintained? If not, then your FOCEXECs could benefit from the FOCUS date utility program discussed in this article.

Calculating the current accounting period is often necessary. For example, your customer needs a second quarter report, and it is currently the third quarter. This is a common situation in financial reporting, as reports must change to fulfill date requirements. To make accounting reports more automatic, create a FOCEXEC called DATES that contains the Dialogue Manager commands shown in Figure 1.

Figure 1 - DATES FOCEXEC

```
-* ****************************************************************
-* Author: Gary C. Browe . Ford Motor Co.      File: DATES
-* Date:  12/89                               Release: 5.8.2
-* System:  Any
-* Master(s): none
-* Function: Date calculations for FOCUS files
-*
-* Remarks:   All &variable dates are in the form MMDDYYYY, using a
-*            4 digit year.  This was done to avoid additional
-*            program logic to calculate dates between the 20th and
-*            21st century.
-*
-* Abbreviations:  REF - reference    MTH - month     BEG - beginning
-*                 CUR - current      YR  - year      END - ending
-*                 ACT - accounting   PRE - previous
-* ****************************************************************
-*
-* ****************************************************************
-* The &variable REFDATE is set outside this FOCEXEC to change the
-* reference date to a date other than the current date.  This is done
-* to fake FOCUS into "thinking" that it is some date other than today.
-* This is useful to "recreate" accounting reports for previous accounting
-* periods or preview future reports without changing program logic.
-* The reference date is typed in MMDDYY format and must refer to a date
-* from 01/01/1950 to 12/31/2049 for this focexec to work correctly.
-* ****************************************************************
-*
-IF &REFDATE.EXIST EQ 0 GOTO SETTODAY ELSE GOTO SETOTHER;
-SETTODAY
```

(continued)

(Figure 1 - DATES FOCEXEC continued)

```
-SET &BASEDATE = &MDY:
-GOTO CONTINUE
-SETOTHER
-SET &BASEDATE = &REFDATE;
-CONTINUE
-*
-* ********************************************************************
-* Current date settings.
-* Note: If the year is between 50 and 99, the date is assumed to be in
-*       the 20th century (1900-1999) else the 21st century (2000-2099)
-* ********************************************************************
-*
-SET &HEADINGDT    = EDIT (&BASEDATE, '99/99/99'):
-SET &MTH          = EDIT (&BASEDATE, '99$$$$'):
-SET &CURMTH       = &MTH;
-SET &YR           = EDIT (&BASEDATE, '$$$$99');
-SET &YR           = IF &YR GE 50 AND &YR LE 99 THEN 19|&YR
-                       ELSE 20|&YR:
-SET &CURYR        = &YR;
-SET &BEGCURMTH    = &MTH | 01 | &CURYR:
-SET &BEGCURYR     = 0101 | &CURYR;
-SET &MIDCURYR     = 0630 | &CURYR:
-SET &ENDCURYR     = 1231 | &CURYR:
-SET &CURQTR       = IF &MTH EQ 1 OR 2 OR 3 THEN 1
-                        ELSE IF &MTH EQ 4 OR 5 OR 6 THEN 2
-                        ELSE IF &MTH EQ 7 OR 8 OR 9 THEN 3
-                        ELSE 4;
-SET &CURQTRNAME   = IF &CURQTR EQ 1 THEN 'FIRST'
-                       ELSE IF &CURQTR EQ 2 THEN 'SECOND'
-                       ELSE IF &CURQTR EQ 3 THEN 'THIRD'
-                       ELSE 'FOURTH':
-*
-* ********************************************************************
-* Accounting Period:  Determine the accounting period which is the
-*                     previous month or quarter.
-* ********************************************************************
-*
-SET &ACTQTR       = IF &CURQTR - 1 EQ 0 THEN 4 ELSE &CURQTR - 1:
-SET &PREQTR       = &ACTQTR:
-SET &ACTQTRNAME   = IF &ACTQTR EQ 1 THEN 'FIRST'
-                       ELSE IF &ACTQTR EQ 2 THEN 'SECOND'
-                       ELSE IF &ACTQTR EQ 3 THEN 'THIRD'
-                       ELSE 'FOURTH':
-SET &PREQTRNAME   = &ACTQTRNAME:
-SET &MTH          = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1:
-SET &MTH          = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &ACTMTH       = &MTH;
-SET &PREMTH       = &ACTMTH:
-SET &YR           = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR:
-SET &ACTYR        = &YR;
-SET &BEGACTMTH    = &MTH | 01 | &ACTYR:
-SET &MIDACTMTH    = &MTH | 15 | &ACTYR:
-SET &BEGPREMTH    = &BEGACTMTH:
-SET &MIDPREMTH    = &MIDACTMTH:
-SET &BEGACTQTR    = IF &ACTQTR EQ 1 THEN 0101 | &ACTYR
-                       ELSE IF &ACTQTR EQ 2 THEN 0401 | &ACTYR
-                       ELSE IF &ACTQTR EQ 3 THEN 0701 | &ACTYR
-                       ELSE 0901 | &ACTYR;
-SET &BEGPREQTR    = &BEGACTQTR:
-SET &ENDACTQTR    = IF &ACTQTR EQ 1 THEN 0331 | &ACTYR
-                       ELSE IF &ACTQTR EQ 2 THEN 0630 | &ACTYR
-                       ELSE IF &ACTQTR EQ 3 THEN 0930 | &ACTYR
-                       ELSE 1231 | &ACTYR;
-SET &ENDPREQTR    = &ENDACTQTR:
```

(continued)

(Figure 1 - DATES FOCEXEC continued)

```
-SET &BEGACTYR      = 0101 | &ACTYR;
-SET &MIDACTYR      = 0630 | &ACTYR;
-SET &ENDACTYR      = 1231 | &ACTYR;
-SET &JAN           = 0101 | &ACTYR;
-SET &FEB           = 0201 | &ACTYR;
-SET &MAR           = 0301 | &ACTYR;
-SET &APR           = 0401 | &ACTYR;
-SET &MAY           = 0501 | &ACTYR;
-SET &JUN           = 0601 | &ACTYR;
-SET &JUL           = 0701 | &ACTYR;
-SET &AUG           = 0801 | &ACTYR;
-SET &SEP           = 0901 | &ACTYR;
-SET &OCT           = 1001 | &ACTYR;
-SET &NOV           = 1101 | &ACTYR;
-SET &DEC           = 1201 | &ACTYR;
-* ****************************************************************
-* Previous year: is the accounting year - 1;
-*
-* Note: If the current month is January, the accounting year will be
-*       the previous year AND the previous year will be the accounting
-*       year - 1.  This is done to allow previous year and accounting
-*       year comparisons in Year End reports.
-* ****************************************************************
-SET &PREYR         = &ACTYR - 1;
-SET &BEGPREYR      = 0101 | &PREYR;
-SET &MIDPREYR      = 0630 | &PREYR;
-SET &ENDPREYR      = 1231 | &PREYR;
-* ****************************************************************
-* Calculate two months ago: which is the current month - 2;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &TWOMTHAGO     = &MTH | 01 | &YR;
-* ****************************************************************
-* Calculate three months ago: which is the current month - 3;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &THREEMTHAGO   = &MTH | 01 | &YR;
-* ****************************************************************
-* Calculate four months ago: which is the current month - 4;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &FOURMTHAGO    = &MTH | 01 | &YR;
-* ****************************************************************
-* Calculate five months ago: which is the current month - 5;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &FIVEMTHAGO    = &MTH | 01 | &YR;
-* ****************************************************************
-* Calculate six months ago: which is the current month - 6;
-* ****************************************************************
-SET &MTH           = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH           = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR            = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &SIXMTHAGO     = &MTH | 01 | &YR;
```

These &variables, when resolved, will contain dates based on the system date or a reference date contained in the report request.

To understand how these &variables are resolved and what values they will contain after execution, execute the TESTDATE FOCEXEC shown in Figure 2 using the ECHO=ALL option (EX TESTDATE ECHO=ALL). The resolved &variables, shown in Figure 3, can then be used in screening statements, DEFINE or COMPUTE statements, column headings, or any other valid statements used in FOCUS.

Figure 2 - TESTDATE FOCEXEC

```
-* ***********************************************************************
-* Author: Gary C. Browe . Ford Motor Co.     File: TESTDATE
-* Date: 11/89                                Release: 5.5.2
-* System: Any
-* Master(s): none
-* Function: Test focexec to type the resolved &variables.
-* ***********************************************************************
-* REFDATE must be typed in MMDDYY format. If the reference date is
-* omitted, the system date will be substituted.
-PROMPT &WHATDATE.I6.TYPE THE REFERENCE DATE IN MMDDYY FORMAT!.
-SET &REFDATE = &WHATDATE;
-INCLUDE DATES
-TYPE HEADINGDT      -   &HEADINGDT
-TYPE CURMTH         -   &CURMTH
-TYPE CURYR          -   &CURYR
-TYPE BEGCURMTH      -   &BEGCURMTH
-TYPE BEGCURYR       -   &BEGCURYR
-TYPE MIDCURYR       -   &MIDCURYR
-TYPE ENDCURYR       -   &ENDCURYR
-TYPE [illegible]    -   &CURQTR
-TYPE CURQTRNAME     -   &CURQTRNAME
-TYPE ACTQTR         -   &ACTQTR
-TYPE PREQTR         -   &PREQTR
-TYPE ACTQTRNAME     -   &ACTQTRNAME
-TYPE PREQTRNAME     -   &PREQTRNAME
-TYPE ACTMTH         -   &ACTMTH
-TYPE PREMTH         -   &PREMTH
-TYPE ACTYR          -   &ACTYR
-TYPE BEGACTMTH      -   &BEGACTMTH
-TYPE MIDACTMTH      -   &MIDACTMTH
-TYPE BEGPREMTH      -   &BEGPREMTH
-TYPE MIDPREMTH      -   &MIDPREMTH
-TYPE BEGACTQTR      -   &BEGACTQTR
-TYPE BEGPREQTR      -   &BEGPREQTR
-TYPE ENDACTQTR      -   &ENDACTQTR
-TYPE ENDPREQTR      -   &ENDPREQTR
-TYPE BEGACTYR       -   &BEGACTYR
-TYPE MIDACTYR       -   &MIDACTYR
-TYPE ENDACTYR       -   &ENDACTYR
-TYPE JAN            -   &JAN
-TYPE FEB            -   &FEB
-TYPE MAR            -   &MAR
-TYPE APR            -   &APR
-TYPE MAY            -   &MAY
-TYPE JUN            -   &JUN
-TYPE JUL            -   &JUL
-TYPE AUG            -   &AUG
-TYPE SEP            -   &SEP
-TYPE OCT            -   &OCT
-TYPE NOV            -   &NOV
-TYPE DEC            -   &DEC
```

(continued)

(Figure 2 - TESTDATE FOCEXEC continued)

```
-TYPE PREYR         =    &PREYR
-TYPE BEGPREYR      =    &BEGPREYR
-TYPE MIDPREYR      =    &MIDPREYR
-TYPE ENDPREYR      =    &ENDPREYR
-TYPE TWOMTHAGO     =    &TWOMTHAGO
-TYPE THREEMTHAGO   =    &THREEMTHAGO
-TYPE FOURMTHAGO    =    &FOURMTHAGO
-TYPE FIVEMTHAGO    =    &FIVEMTHAGO
-TYPE SIXMTHAGO     =    &SIXMTHAGO
-EXIT
```

Figure 3 - Resolved &variables from TESTDATE FOCEXEC

```
HEADINGDT     =   01/01/90
CURMTH        =   01
CURYR         =   1990
BEGCURMTH     =   01011990
BEGCURYR      =   01011990
MIDCURYR      =   06301990
ENDCURYR      =   12311990
CURQTR        =   1
CURQTRNAME    =   FIRST
ACTQTR        =   4
PREQTR        =   4
ACTQTRNAME    =   FOURTH
PREQTRNAME    =   FOURTH
ACTMTH        =   12
PREMTH        =   12
ACTYR         =   1989
BEGACTMTH     =   12011989
MIDACTMTH     =   12151989
BEGPREMTH     =   12011989
MIDPREMTH     =   12151989
BEGACTQTR     =   09011989
BEGPREQTR     =   09011989
ENDACTQTR     =   12311989
ENDPREQTR     =   12311989
BEGACTYR      =   01011989
MIDACTYR      =   06301989
ENDACTYR      =   12311989
JAN           =   01011989
FEB           =   02011989
MAR           =   03011989
APR           =   04011989
MAY           =   05011989
JUN           =   06011989
JUL           =   07011989
AUG           =   08011989
SEP           =   09011989
OCT           =   10011989
NOV           =   11011989
DEC           =   12011989
PREYR         =   1988
BEGPREYR      =   01011988
MIDPREYR      =   06301988
ENDPREYR      =   12311988
TWOMTHAGO     =   11011989
THREEMTHAGO   =   10011989
FOURMTHAGO    =   09011989
FIVEMTHAGO    =   08011989
SIXMTHAGO     =   07011989
```

The DATES procedure is called into the report request via a -INCLUDE statement. The resolved &variables are then available for use (see Figure 4). For more complex reports or special conditions, other Dialogue Manager &variables can be defined using the DATES procedure as a base program (see Figure 5).

When using this technique, bear in mind that a limited number of Dialogue Manager variables are available. This technique makes extensive use of these variables; before using it, ensure that your existing application is not already at or near the limit.

In summary, using this Dialogue Manager DATES procedure throughout your database management system can reduce or eliminate FOCEXEC maintenance during major date changes. It will also have the added benefit of making your FOCEXECs more uniform, thus making them easier to interpret and maintain. ✦

Figure 4 - Sample FOCEXEC

```
-*
-****************************************************************
-* Author: Gary C. Browe , Ford Motor Co.      File: FIGURE4
-* Date:  12/89                                 Release: 5.5.2
-* System:  Any
-* Master(s): EMPLOYEE
-* Function:  Sample example focexec using the DATES include file.
-* Remarks:  The output from this focexec was based on the EMPLOYEE
-*           database supplied with PC/FOCUS release 3.1
-*
-****************************************************************
-SET &REFDATE = 080182 ;
-INCLUDE DATES
-*
-* **************************************************************
-* Define XPAY_DATE as a FOCUS date.
-* **************************************************************
-*
DEFINE FILE EMPLOYEE
XPAY_DATE/MDYY = PAY_DATE:
END
-*
-* **************************************************************
-* Produce the monthly report from the employee database.
-* **************************************************************
-*
TABLE FILE EMPLOYEE
HEADING CENTER
"&HEADINGDT"
"MONTHLY REPORT OF SALARY INCREASE"
"FOR &ACTMTH / &ACTYR </1"
PRINT
 XPAY_DATE AS 'PAY,DATE'
 GROSS     AS 'GROSS,PAY'
BY LAST_NAME  AS 'LAST,NAME'
BY FIRST_NAME AS 'FIRST,NAME'
IF XPAY_DATE GE '&BEGACTMTH'
IF XPAY_DATE LT '&BEGCURMTH'
ON TABLE SUBFOOT
"<46 ------"
"TOTAL GROSS <42 <TOT.GROSS "
END
-EXIT
```

Output

08/01/82
MONTHLY REPORT OF SALARY INCREASE
FOR 07 / 1982

LAST NAME	FIRST NAME	PAY DATE	GROSS PAY
BLACKWOOD	ROSEMARIE	07/30/1982	$1,815.00
CROSS	BARBARA	07/30/1982	$2,256.00
IRVING	JOAN	07/30/1982	$2,238.50
JONES	DIANE	07/30/1982	$1,540.00
MCKNIGHT	ROGER	07/30/1982	$1,342.00
ROMANS	ANTHONY	07/30/1982	$1,760.00
SMITH	MARY	07/30/1982	$1,100.00
STEVENS	ALFRED	07/30/1982	$916.67
TOTAL GROSS			$12,967.17

Figure 5 - Sample FOCEXEC

```
-*
-****************************************************************
-* Author: Gary C. Browe . Ford Motor Co.      File: FIGURE5
                                               Release: 5.6.2
-* Date:  11/89
-* System:  Any
-* Master(s): EMPLOYEE
-* Function:  Example focexec using dates file with additional dates
-*            calculated.
-* Remarks:  The output from this focexec was based on the EMPLOYEE
-*           database supplied with PC/FOCUS release 3.1
-*
-****************************************************************
-SET &REFDATE = 080183 ;
-INCLUDE DATES
-SET &2YRSAGO      = &PREYR - 1;
-SET &BEG2YRSAGO   = 0101 | &2YRSAGO;
-SET &END2YRSAGO   = 1231 | &2YRSAGO;
-*
-* **************************************************************
-* Define XPAY_DATE as a FOCUS date.
-* **************************************************************
-*
DEFINE FILE EMPLOYEE
 XPAY_DATE/MDY = PAY_DATE;
 2YRSAGOPAY/D10.2M = IF XPAY_DATE GE &BEG2YRSAGO
                AND XPAY_DATE LE &END2YRSAGO THEN GROSS ELSE 0;
 PREYRPAY/D10.2M = IF XPAY_DATE GE &BEGPREYR
                AND XPAY_DATE LE &ENDPREYR THEN GROSS ELSE 0;
 ACTYRPAY/D10.2M = IF XPAY_DATE GE &BEGACTYR
                AND XPAY_DATE LE &ENDACTYR THEN GROSS ELSE 0;
END
TABLE FILE EMPLOYEE
HEADING CENTER
 "&HEADINGDT
 "EMPLOYEE PAY HISTORY REPORT
SUM
 2YRSAGOPAY AS '&2YRSAGO,GROSS,EARNINGS'
 PREYRPAY AS '&PREYR,GROSS,EARNINGS'
 ACTYRPAY AS '&ACTYR,GROSS,EARNINGS'
BY LAST_NAME AS 'LAST,NAME'
BY FIRST_NAME AS 'FIRST,NAME'
ON TABLE COLUMN-TOTAL
END
-EXIT
```

Output

```
                         08/01/83
                EMPLOYEE PAY HISTORY REPORT
```

LAST NAME	FIRST NAME	1981 GROSS EARNINGS	1982 GROSS EARNINGS	1983 GROSS EARNINGS
BLACKWOOD	ROSEMARIE	$.00	$1,815.00	$.00
CROSS	BARBARA	$.00	$2,255.00	$.00
IRVING	JOAN	$.00	$2,238.50	$.00
JONES	DIANE	$.00	$1,540.00	$.00
MCKNIGHT	ROGER	$.00	$1,342.00	$.00
ROMANS	ANTHONY	$.00	$1,760.00	$.00
SMITH	MARY	$.00	$1,100.00	$.00
STEVENS	ALFRED	$.00	$916.67	$.00
TOTAL		$.00	$12,967.17	$.00

Gary Browe is a programmer analyst at Ford Motor Company, Parts and Service Division and president of the Detroit-Toledo FOCUS Users Group, TED McFUSE. His experience includes application design, end-user consulting, and giving FOCUS application presentations to FUSE groups. He has been using FOCUS since 1983.

Figure 5 - Sample FOCEXEC

```
-*
-**********************************************************************
-* Author: Gary C. Browe , Ford Motor Co.          File: FIGURES
                                                   Release: 5.5.2
-* Date:  11/89
-* System:  Any
-* Master(s): EMPLOYEE
-* Function:  Example focexec using dates file with additional dates
-*            calculated.
-* Remarks: The output from this focexec was based on the EMPLOYEE
-*          database supplied with PC/FOCUS release 3.1
-*
-**********************************************************************
-SET &REFDATE = 080183 ;
-INCLUDE DATES
-SET &2YRSAGO = &PREYR - 1;
-SET &BEG2YRSAGO = 0101 | &2YRSAGO;
-SET &END2YRSAGO = 1231 | &2YRSAGO;
-*
-**********************************************************************
-* Define XPAY_DATE as a FOCUS date.
-**********************************************************************
-*
DEFINE FILE EMPLOYEE
XPAY_DATE/MDY = PAY_DATE;
2YRSAGOPAY/D10.2M = IF XPAY_DATE GE &BEG2YRSAGO
          AND XPAY_DATE LE &END2YRSAGO THEN GROSS ELSE 0;
PREYRPAY/D10.2M = IF XPAY_DATE GE &BEGPREYR
          AND XPAY_DATE LE &ENDPREYR THEN GROSS ELSE 0;
ACTYRPAY/D10.2M = IF XPAY_DATE GE &BEGACTYR
          AND XPAY_DATE LE &ENDACTYR THEN GROSS ELSE 0;
END
TABLE FILE EMPLOYEE
HEADING CENTER
"&HEADING01"
"EMPLOYEE PAY HISTORY REPORT </1"
SUM
2YRSAGOPAY AS '&2YRSAGO,GROSS,EARNINGS'
PREYRPAY AS '&PREYR,GROSS,EARNINGS'
ACTYRPAY AS '&ACTYR,GROSS,EARNINGS'
BY LAST_NAME AS 'LAST,NAME'
BY FIRST_NAME AS 'FIRST,NAME'
ON TABLE COLUMN-TOTAL
END
[illegible]
```

Output:

```
                       08/03/83
               EMPLOYEE PAY HISTORY REPORT

                                  1981        1982         1983
LAST          FIRST               GROSS       GROSS        GROSS
NAME          NAME                EARNINGS    EARNINGS     EARNINGS

BLACKWOOD     ROSEMARIE           $.00        $1,815.00    $.00
CROSS         BARBARA             $.00        $2,255.00    $.00
IRVING        JOAN                $.00        $2,238.50    $.00
JONES         DIANE               $.00        $1,540.00    $.00
MCKNIGHT      ROGER               $.00        $1,342.00    $.00
ROMANS        ANTHONY             $.00        $1,780.00    $.00
SMITH         MARY                $.00        $1,100.00    $.00
STEVENS       ALFRED              $.00          $916.67    $.00

TOTAL                             $.00       $12,987.17    $.00
```

Gary Browe is a programmer analyst at Ford Motor Company, Parts and Service Division and president of the Detroit-Toledo FOCUS Users Group, TED McFUSE. His experience includes application design, end-user consulting, and giving FOCUS application presentations to FUSE groups. He has been using FOCUS since 1983.

MANAGER

Intelligent Report Maintenance Using Dialogue Manager

"Best Copy Available"

Gary Browe demonstrates a date utility that simplifies maintenance of FOCEXECs that contain specific date periods.

Notice This Material May Be Protected By Copyright Law (Title 17 U.S. Code)

by Gary Browe

Are your reports ready for the year 2000? Do they automatically change when the accounting period changes? Are they flexible and easily maintained? If not, then your FOCEXECs could benefit from the FOCUS date utility program discussed in this article.

Calculating the current accounting period is often necessary. For example, your customer needs a second quarter report, and it is currently the third quarter. This is a common situation in financial reporting, as reports must change to fulfill date requirements. To make accounting reports more automatic, create a FOCEXEC called DATES that contains the Dialogue Manager commands shown in Figure 1.

Figure 1 - DATES FOCEXEC

```
-* ****************************************************************
-* Author: Gary C. Browe . Ford Motor Co.      File: DATES
-* Date:  12/89                                Release: 5.8.2
-* System:  Any
-* Master(s): none
-* Function: Date calculations for FOCUS files
-*
-* Remarks:    All &variable dates are in the form MMDDYYYY, using a
-*             4 digit year.  This was done to avoid additional
-*             program logic to calculate dates between the 20th and
-*             21st century.
-*
-* Abbreviations:  REF - reference    MTH - month     BEG - beginning
-*                 CUR - current      YR  - year      END - ending
-*                 ACT - accounting   PRE - previous
-* ****************************************************************
-*
-* ****************************************************************
-* The &variable REFDATE is set outside this FOCEXEC to change the
-* reference date to a date other than the current date.  This is done
-* to fake FOCUS into "thinking" that it is some date other than today.
-* This is useful to "recreate" accounting reports for previous accounting
-* periods or preview future reports without changing program logic.
-* The reference date is typed in MMDDYY format and must refer to a date
-* from 01/01/1950 to 12/31/2049 for this focexec to work correctly.
-* ****************************************************************
-*
-IF &REFDATE.EXIST EQ 0 GOTO SETTODAY ELSE GOTO SETOTHER;
-SETTODAY
```

(continued)

(Figure 1 - DATES FOCEXEC continued)

```
-SET &BASEDATE = &MDY:
-GOTO CONTINUE
-SETOTHER
-SET &BASEDATE = &REFDATE:
-CONTINUE
-*
-* ***************************************************************
-* Current date settings.
-* Note: If the year is between 50 and 99, the date is assumed to be in
-*       the 20th century (1900-1999) else the 21st century (2000-2099)
-* ***************************************************************
-*
-SET &HEADINGDT   = EDIT (&BASEDATE, '99/99/99'):
-SET &MTH         = EDIT (&BASEDATE, '99$$$$'):
-SET &CURMTH      = &MTH;
-SET &YR          = EDIT (&BASEDATE, '$$$$99');
-SET &YR          = IF &YR GE 50 AND &YR LE 99 THEN 19|&YR
-                      ELSE 20|&YR:
-SET &CURYR       = &YR:
-SET &BEGCURMTH   = &MTH | 01 | &CURYR:
-SET &BEGCURYR    = 0101 | &CURYR;
-SET &MIDCURYR    = 0630 | &CURYR:
-SET &ENDCURYR    = 1231 | &CURYR:
-SET &CURQTR      = IF &MTH EQ 1 OR 2 OR 3 THEN 1
-                       ELSE IF &MTH EQ 4 OR 5 OR 6 THEN 2
-                       ELSE IF &MTH EQ 7 OR 8 OR 9 THEN 3
-                       ELSE 4;
-SET &CURQTRNAME  = IF &CURQTR EQ 1 THEN 'FIRST'
-                     ELSE IF &CURQTR EQ 2 THEN 'SECOND'
-                     ELSE IF &CURQTR EQ 3 THEN 'THIRD'
-                     ELSE 'FOURTH':
-*
-* ***************************************************************
-* Accounting Period:  Determine the accounting period which is the
-*                     previous month or quarter.
-* ***************************************************************
-*
-SET &ACTQTR      = IF &CURQTR - 1 EQ 0 THEN 4 ELSE &CURQTR - 1:
-SET &PREQTR      = &ACTQTR:
-SET &ACTQTRNAME  = IF &ACTQTR EQ 1 THEN 'FIRST'
-                     ELSE IF &ACTQTR EQ 2 THEN 'SECOND'
-                     ELSE IF &ACTQTR EQ 3 THEN 'THIRD'
-                     ELSE 'FOURTH':
-SET &PREQTRNAME  = &ACTQTRNAME:
-SET &MTH         = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1:
-SET &MTH         = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH:
-SET &ACTMTH      = &MTH;
-SET &PREMTH      = &ACTMTH:
-SET &YR          = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR:
-SET &ACTYR       = &YR;
-SET &BEGACTMTH   = &MTH | 01 | &ACTYR:
-SET &MIDACTMTH   = &MTH | 15 | &ACTYR:
-SET &BEGPREMTH   = &BEGACTMTH:
-SET &MIDPREMTH   = &MIDACTMTH:
-SET &BEGACTQTR   = IF &ACTQTR EQ 1 THEN 0101 | &ACTYR
-                     ELSE IF &ACTQTR EQ 2 THEN 0401 | &ACTYR
-                     ELSE IF &ACTQTR EQ 3 THEN 0701 | &ACTYR
-                     ELSE 0901 | &ACTYR;
-SET &BEGPREQTR   = &BEGACTQTR:
-SET &ENDACTQTR   = IF &ACTQTR EQ 1 THEN 0331 | &ACTYR
-                     ELSE IF &ACTQTR EQ 2 THEN 0630 | &ACTYR
-                     ELSE IF &ACTQTR EQ 3 THEN 0930 | &ACTYR
-                     ELSE 1231 | &ACTYR;
-SET &ENDPREQTR   = &ENDACTQTR:
```

(continued)

(Figure 1 - DATES FOCEXEC continued)

```
-SET &BEGACTYR    = 0101 | &ACTYR;
-SET &MIDACTYR    = 0630 | &ACTYR;
-SET &ENDACTYR    = 1231 | &ACTYR;
-SET &JAN         = 0101 | &ACTYR;
-SET &FEB         = 0201 | &ACTYR;
-SET &MAR         = 0301 | &ACTYR;
-SET &APR         = 0401 | &ACTYR;
-SET &MAY         = 0501 | &ACTYR;
-SET &JUN         = 0601 | &ACTYR;
-SET &JUL         = 0701 | &ACTYR;
-SET &AUG         = 0801 | &ACTYR;
-SET &SEP         = 0901 | &ACTYR;
-SET &OCT         = 1001 | &ACTYR;
-SET &NOV         = 1101 | &ACTYR;
-SET &DEC         = 1201 | &ACTYR;
-*************************************************************************
-* Previous year is the accounting year - 1;
-*
-* Note: If the current month is January, the accounting year will be
-*       the previous year AND the previous year will be the accounting
-*       year - 1.  This is done to allow previous year and accounting
-*       year comparisons in Year End reports.
-*************************************************************************
-SET &PREYR       = &ACTYR - 1;
-SET &BEGPREYR    = 0101 | &PREYR;
-SET &MIDPREYR    = 0630 | &PREYR;
-SET &ENDPREYR    = 1231 | &PREYR;
-*************************************************************************
-* Calculate two months ago: which is the current month - 2;
-*************************************************************************
-SET &MTH         = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH         = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR          = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &TWOMTHAGO   = &MTH | 01 | &YR;
-*************************************************************************
-* Calculate three months ago: which is the current month - 3;
-*************************************************************************
-SET &MTH         = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH         = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR          = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &THREEMTHAGO = &MTH | 01 | &YR;
-*************************************************************************
-* Calculate four months ago: which is the current month - 4;
-*************************************************************************
-SET &MTH         = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH         = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR          = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &FOURMTHAGO  = &MTH | 01 | &YR;
-*************************************************************************
-* Calculate five months ago: which is the current month - 5;
-*************************************************************************
-SET &MTH         = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH         = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR          = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &FIVEMTHAGO  = &MTH | 01 | &YR;
-*************************************************************************
-* Calculate six months ago: which is the current month - 6;
-*************************************************************************
-SET &MTH         = IF &MTH - 1 EQ 0 THEN 12 ELSE &MTH - 1;
-SET &MTH         = IF &MTH LT 10 THEN 0 | &MTH ELSE &MTH;
-SET &YR          = IF &MTH EQ 12 THEN &YR - 1 ELSE &YR;
-SET &SIXMTHAGO   = &MTH | 01 | &YR;
```

These &variables, when resolved, will contain dates based on the system date or a reference date contained in the report request.

To understand how these &variables are resolved and what values they will contain after execution, execute the TESTDATE FOCEXEC shown in Figure 2 using the ECHO=ALL option (EX TESTDATE ECHO=ALL). The resolved &variables, shown in Figure 3, can then be used in screening statements, DEFINE or COMPUTE statements, column headings, or any other valid statements used in FOCUS.

Figure 2 - TESTDATE FOCEXEC

```
-* ************************************************************
-* Author: Gary C. Brove . Ford Motor Co.    File: TESTDATE
-* Date: 11/89                               Release: 5.5.2
-* System: Any
-* Master(s): none
-* Function: Test focexec to type the resolved &variables:
-* ************************************************************
-* REFDATE must be typed in MMDDYY [illegible] the reference date
-* omitted, the system date will be substituted.
-PROMPT &WHATDATE.16.TYPE THE REFERENCE DATE IN MMDDYY FORMAT.
-SET &REFDATE = &WHATDATE
-INCLUDE DATES
-TYPE HEADINGDT          =   &HEADINGDT
-TYPE CURMTH             =   &CURMTH
-TYPE CURYR              =   &CURYR
-TYPE BEGCURMTH          =   &BEGCURMTH
-TYPE BEGCURYR           =   &BEGCURYR
-TYPE MIDCURYR           =   &MIDCURYR
-TYPE ENDCURYR           =   &ENDCURYR
-TYPE CUROTR             =   &CUROTR
-TYPE CUROTRNAME         =   &CUROTRNAME
-TYPE ACTOTR             =   &ACTOTR
-TYPE PREOTR             =   &PREOTR
-TYPE ACTOTRNAME         =   &ACTOTRNAME
-TYPE PREOTRNAME         =   &PREOTRNAME
-TYPE ACTMTH             =   &ACTMTH
-TYPE PREMTH             =   &PREMTH
-TYPE ACTYR              =   &ACTYR
-TYPE BEGACTMTH          =   &BEGACTMTH
-TYPE MIDACTMTH          =   &MIDACTMTH
-TYPE BEGPREMTH          =   &BEGPREMTH
-TYPE MIDPREMTH          =   &MIDPREMTH
-TYPE BEGACTOTR          =   &BEGACTOTR
-TYPE BEGPREOTR          =   &BEGPREOTR
-TYPE ENDACTOTR          =   &ENDACTOTR
-TYPE ENDPREOTR          =   &ENDPREOTR
-TYPE BEGACTYR           =   &BEGACTYR
-TYPE MIDACTYR           =   &MIDACTYR
-TYPE ENDACTYR           =   &ENDACTYR
-TYPE JAN                =   &JAN
-TYPE FEB                =   &FEB
-TYPE MAR                =   &MAR
-TYPE APR                =   &APR
-TYPE MAY                =   &MAY
-TYPE JUN                =   &JUN
-TYPE JUL                =   &JUL
-TYPE AUG                =   &AUG
-TYPE SEP                =   &SEP
-TYPE OCT                =   &OCT
-TYPE NOV                =   &NOV
-TYPE DEC                =   &DEC
```

(continued)

(Figure 2 - TESTDATE FOCEXEC continued)

```
-TYPE PREYR          =   &PREYR
-TYPE BEGPREYR       =   &BEGPREYR
-TYPE MIDPREYR       =   &MIDPREYR
-TYPE ENDPREYR       =   &ENDPREYR
-TYPE TWOMTHAGO      =   &TWOMTHAGO
-TYPE THREEMTHAGO    =   &THREEMTHAGO
-TYPE FOURMTHAGO     =   &FOURMTHAGO
-TYPE FIVEMTHAGO     =   &FIVEMTHAGO
-TYPE SIXMTHAGO      =   &SIXMTHAGO
-EXIT
```

Figure 3 - Resolved &variables from TESTDATE FOCEXEC

```
HEADINGDT      =  01/01/90
CURMTH         =  01
CURYR          =  1990
BEGCURMTH      =  01011990
BEGCURYR       =  01011990
MIDCURYR       =  06301990
ENDCURYR       =  12311990
CURQTR         =  1
CURQTRNAME     =  FIRST
ACTQTR         =  4
PREQTR         =  4
ACTQTRNAME     =  FOURTH
PREQTRNAME     =  FOURTH
ACTMTH         =  12
PREMTH         =  12
ACTYR          =  1989
BEGACTMTH      =  12011989
MIDACTMTH      =  12151989
BEGPREMTH      =  12011989
MIDPREMTH      =  12151989
BEGACTQTR      =  09011989
BEGPREQTR      =  09011989
ENDACTQTR      =  12311989
ENDPREQTR      =  12311989
BEGACTYR       =  01011989
MIDACTYR       =  06301989
ENDACTYR       =  12311989
JAN            =  01011989
FEB            =  02011989
MAR            =  03011989
APR            =  04011989
MAY            =  05011989
JUN            =  06011989
JUL            =  07011989
AUG            =  08011989
SEP            =  09011989
OCT            =  10011989
NOV            =  11011989
DEC            =  12011989
PREYR          =  1988
BEGPREYR       =  01011988
MIDPREYR       =  06301988
ENDPREYR       =  12311988
TWOMTHAGO      =  11011989
THREEMTHAGO    =  10011989
FOURMTHAGO     =  09011989
FIVEMTHAGO     =  08011989
SIXMTHAGO      =  07011989
```

The DATES procedure is called into the report request via a -INCLUDE statement. The resolved &variables are then available for use (see Figure 4). For more complex reports or special conditions, other Dialogue Manager &variables can be defined using the DATES procedure as a base program (see Figure 5).

When using this technique, bear in mind that a limited number of Dialogue Manager variables are available. This technique makes extensive use of these variables; before using it, ensure that your existing application is not already at or near the limit.

In summary, using this Dialogue Manager DATES procedure throughout your database management system can reduce or eliminate FOCEXEC maintenance during major date changes. It will also have the added benefit of making your FOCEXECs more uniform, thus making them easier to interpret and maintain. ♦

Figure 4 - Sample FOCEXEC

```
-*
-***********************************************************************
-* Author: Gary C. Browe , Ford Motor Co.     File: FIGURE4
-* Date:  12/89                               Release: 5.5.2
-* System:  Any
-* Master(s): EMPLOYEE
-* Function:  Sample example focexec using the DATES include file.
-* Remarks:  The output from this focexec was based on the EMPLOYEE
-*           database supplied with PC/FOCUS release 3.1
-*
-***********************************************************************
-SET &REFDATE = 080182 ;
-INCLUDE DATES
-*
-* ********************************************************************
-* Define XPAY_DATE as a FOCUS date.
-* ********************************************************************
-*
DEFINE FILE EMPLOYEE
XPAY_DATE/MDYY = PAY_DATE;
END
-*
-* ********************************************************************
-* Produce the monthly report from the employee database.
-* ********************************************************************
-*
TABLE FILE EMPLOYEE
HEADING CENTER
"&HEADMEDT"
"MONTHLY REPORT OF SALARY INCREASE"
"FOR &ACTMTH / &ACTYR </1"
PRINT
 XPAY_DATE AS 'PAY,DATE'
 GROSS     AS 'GROSS,PAY'
BY LAST_NAME  AS 'LAST,NAME'
BY FIRST_NAME AS 'FIRST,NAME'
IF XPAY_DATE GE '&BEGACTMTH'
IF XPAY_DATE LT '&BEGCURMTH'
ON TABLE SUBFOOT
"<46 ______"
"TOTAL GROSS <42 <TOT.GROSS "
END
-EXIT
```

Output

08/01/82
MONTHLY REPORT OF SALARY INCREASE
FOR 07 / 1982

LAST NAME	FIRST NAME	PAY DATE	GROSS PAY
BLACKWOOD	ROSEMARIE	07/30/1982	$1,815.00
CROSS	BARBARA	07/30/1982	$2,255.00
IRVING	JOAN	07/30/1982	$2,238.50
JONES	DIANE	07/30/1982	$1,540.00
MCKNIGHT	ROGER	07/30/1982	$1,342.00
ROMANS	ANTHONY	07/30/1982	$1,760.00
SMITH	MARY	07/30/1982	$1,100.00
STEVENS	ALFRED	07/30/1982	$916.67
TOTAL GROSS			$12,967.17

Figure 5 - Sample FOCEXEC

```
-*
-*****************************************************************
-* Author: Gary C. Browe . Ford Motor Co.     File: FIGURES
-* Date:   11/89                               Release: 5.6.2
-* System:  Any
-* Master(s): EMPLOYEE
-* Function:  Example focexec using dates file with additional dates
-*            calculated.
-* Remarks:  The output from this focexec was based on the EMPLOYEE
-*           database supplied with PC/FOCUS release 3.1
-*
-*****************************************************************
-SET &REFDATE = 080183 ;
-INCLUDE DATES
-SET &2YRSAGO      = &PREYR - 1 ;
-SET &BEG2YRSAGO   = 0101 | &2YRSAGO ;
-SET &END2YRSAGO   = 1231 | &2YRSAGO ;
-*
-*****************************************************************
-* Define XPAY_DATE as a FOCUS date
-*****************************************************************
-*
DEFINE FILE EMPLOYEE
 XPAY_DATE/MDY = PAY_DATE ;
 2YRSAGOPAY/D10.2M = IF XPAY_DATE GE &BEG2YRSAGO
                 AND XPAY_DATE LE &END2YRSAGO THEN GROSS ELSE 0;
 PREYRPAY/D10.2M = IF XPAY_DATE GE &BEGPREYR
                 AND XPAY_DATE LE &ENDPREYR THEN GROSS ELSE 0;
 ACTYRPAY/D10.2M = IF XPAY_DATE GE &BEGACTYR
                 AND XPAY_DATE LE &ENDACTYR THEN GROSS ELSE 0;
END
TABLE FILE EMPLOYEE
HEADING CENTER
"&HEADINGDT
"EMPLOYEE PAY HISTORY REPORT
SUM
 2YRSAGOPAY AS '&2YRSAGO,GROSS,EARNINGS'
 PREYRPAY AS '&PREYR,GROSS,EARNINGS'
 ACTYRPAY AS '&ACTYR,GROSS,EARNINGS'
BY [illegible] AS 'LAST,NAME'
BY FIRST_NAME AS 'FIRST,NAME'
ON TABLE COLUMN-TOTAL
END
-EXIT
```

Output

08/01/83
EMPLOYEE PAY HISTORY REPORT

LAST NAME	FIRST NAME	1981 GROSS EARNINGS	1982 GROSS EARNINGS	1983 GROSS EARNINGS
BLACKWOOD	ROSEMARIE	$.00	$1,815.00	$.00
CROSS	BARBARA	$.00	$2,255.00	$.00
IRVING	JOAN	$.00	$2,238.50	$.00
JONES	DIANE	$.00	$1,540.00	$.00
MCKNIGHT	ROGER	$.00	$1,342.00	$.00
ROMANS	ANTHONY	$.00	$1,760.00	$.00
SMITH	MARY	$.00	$1,100.00	$.00
STEVENS	ALFRED	$.00	$916.67	$.00
TOTAL		$.00	$12,967.17	$.00

Gary Browe is a programmer analyst at Ford Motor Company, Parts and Service Division and president of the Detroit-Toledo FOCUS Users Group, TED McFUSE. His experience includes application design, end-user consulting, and giving FOCUS application presentations to FUSE groups. He has been using FOCUS since 1983.

Figure 5 - Sample FOCEXEC

```
-*
-****************************************************************************
-* Author: Gary C. Browe , Ford Motor Co.        File: FIGURE5
-* Date:   11/89                                 Release: 5.5.2
-* System:  Any
-* Master(s): EMPLOYEE
-* Function:  Example focexec using dates file with additional dates
-*            calculated.
-* Remarks:  The output from this focexec was based on the EMPLOYEE
-*           database supplied with PC/FOCUS release 5.1
-*
-****************************************************************************
-SET &REFDATE = 080183 ;
-INCLUDE DATES
-SET &2YRSAGO = &PREYR - 1;
-SET &BEG2YRSAGO = 0101 | &2YRSAGO;
-SET &END2YRSAGO = 1231 | &2YRSAGO;
-*
-****************************************************************************
-* Define XPAY_DATE as a FOCUS date.
-****************************************************************************
-*
DEFINE FILE EMPLOYEE [illegible]
XPAY_DATE/YMD = PAY_DATE;
2YRSAGOPAY/D10.2M = IF XPAY_DATE GE '&BEG2YRSAGO'
                AND XPAY_DATE LE '&END2YRSAGO' THEN GROSS ELSE 0;
PREYRPAY/D10.2M = IF XPAY_DATE GE '&BEGPREYR'
                AND XPAY_DATE LE '&ENDPREYR' THEN GROSS ELSE 0;
ACTYRPAY/D10.2M = IF XPAY_DATE GE '&BEGACTYR'
                AND XPAY_DATE LE '&ENDACTYR' THEN GROSS ELSE 0;
END
TABLE FILE EMPLOYEE
HEADING CENTER
"&HEADING01"
"EMPLOYEE PAY HISTORY REPORT </1"
SUM
 2YRSAGOPAY AS '&2YRSAGO,GROSS,EARNINGS'
 PREYRPAY AS '&PREYR,GROSS,EARNINGS'
 ACTYRPAY AS '&ACTYR,GROSS,EARNINGS'
BY LAST_NAME AS 'LAST,NAME'
BY FIRST_NAME AS 'FIRST,NAME'
ON TABLE [illegible]
END
[illegible]
```

Output:

```
[illegible]                  08/03/83
                    EMPLOYEE PAY HISTORY REPORT
```

LAST NAME	FIRST NAME	1981 GROSS EARNINGS	1982 GROSS EARNINGS	1983 GROSS EARNINGS
BLACKWOOD	ROSEMARIE	$.00	$1,815.00	$.00
CROSS	BARBARA	$.00	$2,255.00	$.00
IRVING	JOAN	$.00	$2,238.50	$.00
JONES	DIANE	$.00	$1,540.00	$.00
MCKNIGHT	ROGER	$.00	$1,342.00	$.00
ROMANS	ANTHONY	$.00	$1,760.00	$.00
SMITH	MARY	$.00	$1,100.00	$.00
STEVENS	ALFRED	$.00	$916.67	$.00
TOTAL		$.00	$12,967.17	$.00

Gary Browe is a programmer analyst at Ford Motor Company, Parts and Service Division and president of the Detroit-Toledo FOCUS Users Group, TED McFUSE. His experience includes application design, end-user consulting, and giving FOCUS application presentations to FUSE groups. He has been using FOCUS since 1983.

Star Byte Systems, Inc.
1819 Lindbergh Lane
Daytona Beach, Florida 32124
(904) 756-3963

FAX TRANSMITTAL SHEET

DATE: FEBRUARY 25, 2000

COMPANY: US PATENT OFFICE FAX NO: (703) 308-6916
ATTENTION: GERALD DOST TIME SENT: 10:45 AM

FROM: ______ FAX NO: ______
RE: RE-EXAM OF PATENT # 5806063, (BRUCE DICKENS)

NUMBER OF PAGES (INCLUDING COVER SHEET): 6

COMMENTS: PAGE 2 - MY 1989 COPYRIGHT
PAGE 3 - ONE SAMPLE OF MY SOURCE CODE
AND VERBAL DESCRIPTION
PAGES 4 → 6 - MORE EXAMPLES OF THE
PERTINENT CODE THROUGHOUT MY
PROGRAM.

FAX RECEIVED
FEB 2 5 2000
PETITIONS OFFICE

CERTIFICATE OF COPYRIGHT REGISTRATION

UNITED STATES COPYRIGHT OFFICE · THE LIBRARY OF CONGRESS

OFFICIAL SEAL

This certificate, issued under the seal of the Copyright Office in accordance with the provisions of section 410(a) of title 17, United States Code, attests that copyright registration has been made for the work identified below. The information in this certificate has been made a part of the Copyright Office records.

REGISTER OF COPYRIGHTS
United States of America

FORM TX
UNITED STATES COPYRIGHT OFFICE

REGISTRATION NUMBER
TXu 869-241

TX TXU

EFFECTIVE DATE OF REGISTRATION
APR 20 1989
Month Day Year

DO NOT WRITE ABOVE THIS LINE. IF YOU NEED MORE SPACE, USE A SEPARATE CONTINUATION SHEET.

1

TITLE OF THIS WORK ▼
Dentist Office I

PREVIOUS OR ALTERNATIVE TITLES ▼

PUBLICATION AS A CONTRIBUTION If this work was published as a contribution to a periodical, serial, or collection, give information about the collective work in which the contribution appeared. Title of Collective Work ▼

If published in a periodical or serial give: Volume ▼ Number ▼ Issue Date ▼ On Pages ▼

2

NAME OF AUTHOR ▼
a Paul A. Graffeo

DATES OF BIRTH AND DEATH
Year Born ▼ 1940 Year Died ▼

Was this contribution to the work a "work made for hire"? ☐ Yes ☒ No

AUTHOR'S NATIONALITY OR DOMICILE
Name of Country
OR { Citizen of ▶ U.S.A. / Domiciled in ▶

WAS THIS AUTHOR'S CONTRIBUTION TO THE WORK
Anonymous? ☒ Yes ☐ No
Pseudonymous? ☐ Yes ☐ No
If the answer to either of these questions is "Yes," see detailed instructions.

NOTE

NATURE OF AUTHORSHIP Briefly describe nature of the material created by this author in which copyright is claimed. ▼
Author of entire text of user's manual and computer program text

NAME OF AUTHOR ▼
b

DATES OF BIRTH AND DEATH
Year Born ▼ Year Died ▼

Was this contribution to the work a "work made for hire"? ☐ Yes ☐ No

AUTHOR'S NATIONALITY OR DOMICILE
Name of country
OR { Citizen of ▶ / Domiciled in ▶

WAS THIS AUTHOR'S CONTRIBUTION TO THE WORK
Anonymous? ☐ Yes ☐ No
Pseudonymous? ☐ Yes ☐ No
If the answer to either of these questions is "Yes," see detailed instructions.

NATURE OF AUTHORSHIP Briefly describe nature of the material created by this author in which copyright is claimed. ▼

NAME OF AUTHOR ▼
c

DATES OF BIRTH AND DEATH
Year Born ▼ Year Died ▼

Was this contribution to the work a "work made for hire"? ☐ Yes ☐ No

AUTHOR'S NATIONALITY OR DOMICILE
Name of Country
OR { Citizen of ▶ / Domiciled in ▶

WAS THIS AUTHOR'S CONTRIBUTION TO THE WORK
Anonymous? ☐ Yes ☐ No
Pseudonymous? ☐ Yes ☐ No
If the answer to either of these questions is "Yes," see detailed instructions.

NATURE OF AUTHORSHIP Briefly describe nature of the material created by this author in which copyright is claimed. ▼

3

YEAR IN WHICH CREATION OF THIS WORK WAS COMPLETED This information must be given in all cases.
1989 ◀ Year

DATE AND NATION OF FIRST PUBLICATION OF THIS PARTICULAR WORK
Complete this information ONLY if this work has been published. Month ▶ Day ▶ Year ▶ ◀ Nation

4

COPYRIGHT CLAIMANT(S) Name and address must be given even if the claimant is the same as the author given in space 2.▼
Star Byte Systems, Inc.
17 Archer Drive
Stony Brook, New York 11790

See instructions before completing this space.

TRANSFER If the claimant(s) named here in space 4 are different from the author(s) named in space 2, give a brief statement of how the claimant(s) obtained ownership of the copyright.▼
by Agreement

DO NOT WRITE HERE / OFFICE USE ONLY
APPLICATION RECEIVED APR 20 1989
ONE DEPOSIT RECEIVED APR 20 1989 *
TWO DEPOSITS RECEIVED
REMITTANCE NUMBER AND DATE

MORE ON BACK ▶ • Complete all applicable spaces (numbers 5-11) on the reverse side of this page. • See detailed instructions. • Sign the form at line 10.

DO NOT WRITE HERE

THESE LINES FOUND IN APLYPAT

```
5812 IF VAL(RIGHT$(SVDATE$(N%),2))>30 THEN Y1$="19" ELSE Y1$="20"
5815 MASK$=LEFT$(SVDATE$(N%),2)+"-"+MID$(SVDATE$(N%),3,2)+"-"+Y1$+RIGHT$(SVDATE$
(N%),2)
```

THESE TWO LINES OF CODE ARE THE HEART OF MY Y2K SOLUTION. THIS WAS WRITTEN IN 1988.

IT SAYS THAT IF THE VALUE OF THE TWO DATE DIGITS SAVED IS GREATER THAN 30, THEN PUT A "19" IN FRONT OF IT, OTHERWISE, PUT A "20" IN THE FRONT

I.E. 1982 OR 2012

DENTIST OFFICE I (c) 1989 Star Byte Systems, Inc. All rights reserved
by P.A. Graffeo

CHRGPAT.LST

```
51600 ' ----------- convert to show$ -----------------------
51610 IF VAL(RIGHT$(CHRG$(N%,1),2))>30 THEN Y1$="19" ELSE Y1$="20"
51620 SHOW$(1)=LEFT$(CHRG$(N%,1),2)+"-"+MID$(CHRG$(N%,1),3,2)+"-"+Y1$+RIGHT$(CHR
G$(N%,1),2)
```

APLYPAT.LST

```
5812 IF VAL(RIGHT$(SVDATE$(N%),2))>30 THEN Y1$="19" ELSE Y1$="20"
5815 MASK$=LEFT$(SVDATE$(N%),2)+"-"+MID$(SVDATE$(N%),3,2)+"-"+Y1$+RIGHT$(SVDATE$
(N%),2)

51600 ' ---- convert to show$
51605 IF VAL(RIGHT$(PYMT$(N%,1),2))>30 THEN Y1$="19" ELSE Y1$="20"
51610 SHOW$(1)=LEFT$(PYMT$(N%,1),2)+"-"+MID$(PYMT$(N%,1),3,2)+"-"+Y1$+RIGHT$(PYM
T$(N%,1),2)

47045    MO=VAL(MID$(SEARCH$(TOP%),8,2)):DA=VAL(MID$(SEARCH$(TOP%),10,2)):YR=VAL(
MID$(SEARCH$(TOP%),12,2))
47047    IF YR<31 THEN YR=YR+100
47050    FILEDAY=INT((365.25*(YR-1))+(30.43*(MO-1))+DA)

47240    MO=VAL(MID$(SEARCH$(Y%),8,2)):DA=VAL(MID$(SEARCH$(Y%),10,2)):YR=VAL(MID$
(SEARCH$(Y%),12,2))
47245    IF YR<31 THEN YR=YR+100
47250    FILEDAY=INT((365.25*(YR-1))+(30.43*(MO-1))+DA)

47360    MO=VAL(MID$(TEMP$,8,2)):DA=VAL(MID$(TEMP$,10,2)):YR=VAL(MID$(TEMP$,12,2)
)
47370    IF YR<31 THEN YR=YR+100
47380    FILEDAY=INT((365.25*(YR-1))+(30.43*(MO-1))+DA)
```

REVUPAT.LST

```
51050 'rem ---- figure debit age ----
51055 MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGHT$(H
OLDING$(1),2))
51060 IF YR<31 THEN YR=YR+100
51065 PAST=(365.25*(YR-1))+(30.43*(MO-1))+DA
```

BILLPAT.LST

```
7330     MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGHT
$(HOLDING$(1),2))
7340     IF YR<31 THEN YR=YR+100
7350     DAY=INT((365.25*(YR-1))+(30.43*(MO-1))+DA)

46330     MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGH
T$(HOLDING$(1),2))
46335     IF YR<31 THEN YR=YR+100
46340     DAY=INT((365.25*(YR-1))+(30.43*(MO-1))+DA)
```

TALYPAT.LST

```
5690      MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGHT
$(HOLDING$(1),2))
5700      IF YR<31 THEN YR=YR+100

6120      MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGHT
$(HOLDING$(1),2))
6130      IF YR<31 THEN YR=YR+100

10510     MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGH
T$(HOLDING$(1),2))
10520     IF YR<31 THEN YR=YR+100

15480     MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGH
T$(HOLDING$(1),2))
15490     IF YR<31 THEN YR=YR+100

47100     MX=VAL(LEFT$(INFO$(66),2)):DX=VAL(MID$(INFO$(66),3,2)):YX=VAL(RIGHT$(I
NFO$(66),2))
47105     IF YX<31 THEN YX=YX+100

47800     MX=VAL(LEFT$(INFO$(66),2)):DX=VAL(MID$(INFO$(66),3,2)):YX=VAL(RIGHT$(I
NFO$(66),2))
47805     IF YX<31 THEN YX=YX+100
```

BUSIPAT.LST

```
14715   MO=VAL(LEFT$(CKPMT$,2)):DA=VAL(MID$(CKPMT$,4,2)):YR=VAL(RIGHT$(CKPMT$,2)
):IF RIGHT$(CKPMT$,2)<>"  " THEN IF YR<31 THEN YR=YR+100
14720   P1=(365.25*(YR-1))+(30.43*(MO-1))+DA
14725   MO=VAL(LEFT$(LSTPMT$,2)):DA=VAL(MID$(LSTPMT$,4,2)):YR=VAL(RIGHT$(LSTPMT$
,2)):IF RIGHT$(LSTPMT$,2)<>"  " THEN IF YR<31 THEN YR=YR+100
14730   P2=(365.25*(YR-1))+(30.43*(MO-1))+DA

17700     MO=VAL(LEFT$(INFO$(66),2)):DA=VAL(MID$(INFO$(66),3,2)):YR=VAL(RIGHT$(I
NFO$(66),2))
17710     IF YR<31 THEN YR=YR+100

18100     MO=VAL(LEFT$(INFO$(66),2)):DA=VAL(MID$(INFO$(66),3,2)):YR=VAL(RIGHT$(I
NFO$(66),2))
18110     IF YR<31 THEN YR=YR+100

20720     MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGH
T$(HOLDING$(1),2))
20725     IF YR<31 THEN YR=YR+100

47050   MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGHT$
(HOLDING$(1),2))
47055   IF YR<31 THEN YR=YR+100

47200 ' ------------- figure inactive period --------------
47210 OKAY=0
47220 IF INFO$(63)<>"        " THEN LSTDATE$=INFO$(63) ELSE LSTDATE$=INFO$(66)
47230 MO2=VAL(LEFT$(LSTDATE$,2)):DA2=VAL(MID$(LSTDATE$,3,2)):YR2=VAL(RIGHT$(LSTD
ATE$,2))
47240 IF YR2<31 THEN YR2=YR2+100 '*
47250 LSTDAY=INT((365.25*(YR2-1))+(30.43*(MO2-1))+DA2+PERIOD)
47260 IF LSTDAY>=INT(TODAY) THEN RETURN
```

INSRPAT.LST

```
7040 IF VAL(MID$(RECORD$(1),17,2))>30 THEN Y1$="19" ELSE Y1$="20" '*
7045 SHOW$(1)=MID$(RECORD$(1),11,6)+Y1$+MID$(RECORD$(1),17,2)

51150 ' ---- compute time ----
51160 MO=VAL(MID$(TEMPREC$(I%),6,2)):DA=VAL(MID$(TEMPREC$(I%),8,2)):YR=VAL(MID$(
TEMPREC$(I%),10,2))
51165 IF YR<31 THEN YR=YR+100
51170 DAY=INT((365.25*(YR-1))+(30.43*(MO-1))+DA)
```

CLAIMREV.LST

```
8600    MO=VAL(MID$(STORE$,8,2)):DA=VAL(MID$(STORE$,10,2)):YR=VAL(MID$(STORE$,12,
2))
8610    IF YR<31 THEN YR=YR+100

10925    MO=VAL(MID$(STORE$,8,2)):DA=VAL(MID$(STORE$,10,2)):YR=VAL(MID$(STORE$,12
,2))
10930    IF YR<31 THEN YR=YR+100

51100 MO1=VAL(LEFT$(DT1$,2)):DA1=VAL(MID$(DT1$,3,2)):YR1=VAL(RIGHT$(DT1$,2))
51105 IF YR1<31 THEN YR1=YR1+100
51110 STDAY=(365.25*(YR1-1))+(30.43*(MO1-1))+DA1
51120 MO2=VAL(LEFT$(DT2$,2)):DA2=VAL(MID$(DT2$,3,2)):YR2=VAL(RIGHT$(DT2$,2))
51125 IF YR2<31 THEN YR2=YR2+100
51130 FINDAY=(365.25*(YR2-1))+(30.43*(MO2-1))+DA2
```

STATPAT.LST

```
6130      MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGHT
$(HOLDING$(1),2))
6140      IF YR<31 THEN YR=YR+100

15730      MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGH
T$(HOLDING$(1),2))
15735      IF YR<31 THEN YR=YR+100

18530      MO=VAL(LEFT$(HOLDING$(1),2)):DA=VAL(MID$(HOLDING$(1),3,2)):YR=VAL(RIGH
T$(HOLDING$(1),2))
18535      IF YR<31 THEN YR=YR+100
```

****
*** ACTIVITY REPORT ***
****

RECEPTION OK

TX/RX NO.	9770
CONNECTION TEL	
CONNECTION ID	
START TIME	02/25 11:45
USAGE TIME	03'24
PAGES	6
RESULT	OK

Chronology of Dickens Patent (5,806,063) prosecution

10/3/96 **Filing Date of the application.**

11/17/97 **Mail Date of the first office action.**

Rejections in case:

1. 35 U.S.C. 102(e) rejection under "The Year 200 and 2-Digit Dates: A Guide for Planning and Implementation, Third Edition, May 1996."
2. 35 U.S.C. 112, first paragraph rejection. Examiner alleges that the claims increase the number of date digits, which is inconsistent with the specification, which indicate that the invention "solves the Y2K problem without introducing additional digits."

3/20/98 **Response to first office action filed.**

Key point in response: "dates are temporarily converted and reformatted...Instead, the original dates in data storage remain undisturbed...This aspect of the invention thus allows conversion of dates to compensate for century designations without requiring the addition of data fields to permanently store the century designations." Applicant overcomes 35 U.S.C 112 by explanation of invention.

Second Key point: Applicant swears behind May 1996 reference with a Declaration under 37 C.F.R. 131.

4/2/98 **Telephone interview conducted.**

Key point in the interview: Examiner indicated possible allowance of case if claim was amended to indicate that "the use of additional century digits did not include their storage in the database."

4/2/98 **Supplemental response filed.**

Key point in response: "Claims 1 and 11 have now been amended so as to not require storage of the converted dates... the claimed invention does not require additional data fields for storage."

4/8/98 **Notice of allowance mailed.**

Key point: Examiner's reason for allowance stresses two main points, namely, that the prior art does not disclose "threshold year digits as used to determine a pair of century digits to be used for computation, but without enlarging the number date digits of the database."

EXAMINER'S ANALYSIS OF USPAT 5,806,063 CLAIM 1

A method of processing symbolic representations of dates stored in a **database**, comprising the steps of:	There are a variety of Y2K problems, of which one is to use a database that contains 2-digit date fields. These must be **recognized**, not a trivial problem in some cases. This differs from **correction** of a database by either modification or creation of a new database, and both problems are related to but distinct from dealing with the software rather than the database.
providing a database with symbolic representations of dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the **symbolic representations** of dates falling within a 10-decade period of time;	The parameters, such as M_1M_2, of which there are 5 pairs in the claim, must be addressed in some fashion in any valid reference. Many of the general discussions fail to do so explicitly. For instance, a potential reference may simply interpret the 2-digit years in a range as inherently being 20th century years, without forming a symbolic representation that includes C_1C_2 as indicated below.
selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the **earliest Y_1Y_2 year designator** in the database;	Selecting here is interpreted as **searching** for the earliest 2-digit year in the database, in light of col 1 lines 51-56 of the SUMMARY, where the invention is applied in a manner which "requires no user input".
determining a **century designator C_1C_2** for each symbolic representation of a date in the database, C_1C_2 **having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B**; and	The application of C_1C_2 is important because it allows the database dates to be sorted properly. The use of Y_AY_B determined by the search for the earliest date **differs significantly from the use of the system clock to determine this value, as is done in LISP and proposed in various references.**
reformatting the symbolic representation of the date with the values C_1C_2, Y_1Y_2, M_1M_2 and D_1D_2 **to facilitate further processing of the dates.**	Further processing includes **sorting** with the century digits **in the leading position**, as noted at col 2, see col 2 lines 16-21.

5806063

UTILITY SERIAL NUMBER 08/725574

SEP 08 1998

PATENT NUMBER

SERIAL NUMBER	FILING DATE	CLASS	SUBCLASS	GROUP ART UNIT	EXAMINER
08/725,574	10/03/96	~~395~~ 707	611 6	~~2309~~ 2307	Amsbury

APPLICANTS

BRUCE DICKENS, IRVINE, CA.

2171

CONTINUING DATA*******************

VERIFIED None

FOREIGN/PCT APPLICATIONS**********

VERIFIED None

FOREIGN FILING LICENSE GRANTED 12/07/96

Foreign priority claimed / 35 USC 119 conditions met / Verified and Acknowledged	AS FILED	STATE OR COUNTRY	SHEETS DRWGS.	TOTAL CLAIMS	INDEP. CLAIMS	FILING FEE RECEIVED	ATTORNEY'S DOCKET NO.
☐ yes ☒ no / ☐ yes ☒ no / Examiner's Initials		CA	1	15	2	$770.00	11151

ADDRESS

~~GREGORY D GARMONG~~
~~PO BOX 12460~~
~~ZEPHYR COVE NV 89448~~

G. [illegible] Gosnell
[illegible] Seltzer, Park + Gibson, P.A.
[illegible] Drawer 34009
Charlotte, NC 28234

TITLE

DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

U.S. DEPT. OF COMM./PAT. & TM—PTO-436L (Rev.12-94)

CERTIFICATE OF CORRECTION

DEC 29 1998

PARTS OF APPLICATION FILED SEPARATELY

Applications Examiner

NOTICE OF ALLOWANCE MAILED

4-8-98

Assistant Examiner

CLAIMS ALLOWED

Total Claims	Print Claim
15	1

ISSUE FEE

Amount Due	Date Paid
1320.00	5-22-98

WAYNE AMSBURY

DRAWING

Sheets Drwg.	Figs. Drwg.	Print Fig.
1	2	2

PTO UTILITY GRANT

Paper Number 12

The United States of America

The Commissioner of Patents and Trademarks

Has received an application for a patent for a new and useful invention. The title and description of the invention are enclosed. The requirements of law have been complied with, and it has been determined that a patent on the invention shall be granted under the law.

Therefore, this

United States Patent

Grants to the person(s) having title to this patent the right to exclude others from making, using, offering for sale, or selling the invention throughout the United States of America or importing the invention into the United States of America for the term set forth below, subject to the payment of maintenance fees as provided by law.

If this application was filed prior to June 8, 1995, the term of this patent is the longer of seventeen years from the date of grant of this patent or twenty years from the earliest effective U.S. filing date of the application, subject to any statutory extension.

If this application was filed on or after June 8, 1995, the term of this patent is twenty years from the U.S. filing date, subject to an statutory extension. If the application contains a specific reference to an earlier filed application or applications under 35 U.S.C. 120, 121 or 365(c), the term of the patent is twenty years from the date on which the earliest application was filed, subject to any statutory extension.

Commissioner of Patents and Trademarks

Attest

Form PTO-1584 (Rev. 2/97)

Staple Issue Slip Here

POSITION		ID NO.	DATE
CLASSIFIER	#44		11-5-96
EXAMINER		287	12/6
TYPIST		513	12-?
VERIFIER		703	12/9/96
CORPS CORR.			
SPEC. HAND			
FILE MAINT.			
DRAFTING			

INDEX OF CLAIMS

Claim Final	Claim Original	Date 11/6/97	Date 4/2/98						
1	1	✓	=						
2	2								
3	3								
4	4								
5	5								
6	6								
7	7								
10	8								
8	9								
9	10								
11	11								
12	12								
15	13								
13	14								
14	15	✓	=						
	16								
	17								
	18								
	19								
	20								
	21								
	22								
	23								
	24								
	25								
	26								
	27								
	28								
	29								
	30								
	31								
	32								
	33								
	34								
	35								

Claim Final	Claim Original	Date							
	51								
	52								
	53								
	54								
	55								
	56								
	57								
	58								
	59								
	60								
	61								
	62								
	63								
	64								
	65								
	66								
	67								
	68								
	69								
	70								
	71								
	72								
	73								
	74								
	75								
	76								
	77								
	78								
	79								
	80								
	81								
	82								
	83								
	84								
	85								
	86								

SYMBOLS

✓	Rejected
=	Allowed
- (Through numeral)	Canceled
÷	Restricted
N	Non-elected
I	Interference
A	Appeal
O	Objected

PATENT APPLICATION SERIAL NO. 08/725574

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

250 LC 10/18/96 08725574
1 101 770.00 CK

PTO-1556
(5/87)

08/725574

MAIL ROOM
OCT 4 3 1996
PAT. & TRADEMARK OFF.

FORM PTO-1082

Case Docket No. 11151

THE COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

Sir:
Transmitted herewith for filing is the patent application of
Inventor: Bruce Dickens
For: DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

Enclosed are:
XX 1 sheets of drawing.
XX An assignment of the invention to McDonnell Douglas Corporation

☐ A certified copy of a ________ application.
☐ An associate power of attorney.
☐ A verified statement to establish small entity status under 37 CFR 1.9 and 37 CFR 1.27.
☒ Exhibit A

The filing fee has been calculated as shown below:

FOR:	NO. FILED (Col. 1)	NO. EXTRA (Col. 2)
BASIC FEE		
TOTAL CLAIMS	15 -20 =	* 0
INDEP CLAIMS	2 - 3 =	* 0
☐ MULTIPLE DEPENDENT CLAIM PRESENTED		

* If the difference in Col. 1 is less than zero, enter "0" in Col. 2

SMALL ENTITY RATE	SMALL ENTITY FEE		OTHER THAN A SMALL ENTITY RATE	OTHER THAN A SMALL ENTITY FEE
	$ 170	OR		$ ~~340~~ 770
X 6 =	$	OR	X12 =	$
X17 =	$	OR	X34 =	$
+55 =	$	OR	+110 =	$
Assign.	$	OR	Assign.	$ 40
TOTAL			TOTAL	810

☐ Please charge my Deposit Account No. ________ the amount of $________. A duplicate copy of this sheet is enclosed.

XX A check in the amount of $ 810 to cover the filing fee is enclosed.

XX The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 07-0143. A duplicate copy of this sheet is enclosed.

☒ Any additional filing fees required under 37 CFR 1.16.
XX Any patent application processing fees under 37 CFR 1.17.

XX The Commissioner is hereby authorized to charge payment of the following fees during the pendency of this application or credit any overpayment to Deposit Account No. 07-0143. A duplicate copy of this sheet is enclosed.

☒ Any patent application processing fees under 37 CFR 1.17.
☐ The issue fee set in 37 CFR 1.18 at or before mailing of the Notice of Allowance, pursuant to 37 CFR 1.311 (b).
☐ Any filing fees under 37 CFR 1.16 for presentation of extra claims.
X Any deficiencies in fees in this application.

Gregory Garmong
P.O. Box 12460
Zephyr Cove, NV 89448

Respectfully submitted,

Gregory Garmong, Reg. No. 29,382

1 7 101A

08725574

MAIL ROOM
OCT 3 1996
4
PAT. & TRADEMARK OFF.

APPLICATION OF

BRUCE DICKENS

for

UNITED STATES PATENT

on

DATE FORMATTING AND SORTING FOR DATES
SPANNING THE TURN OF THE CENTURY

Docket No. 11151

MAIL ROOM
OCT 3 1996
4
PAT. & TRADEMARK OFF.

DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

BACKGROUND OF THE INVENTION

This invention relates to the manipulation of information in a database, and, in particular, to the determination of dates in a useful form.

Dates are stored as symbolic representations in computer databases in varying formats. For example, a date may be represented in the numerical representation MM/DD/YY, where MM is a two-digit month designator, DD is a two-digit day designator, and YY is a two-digit year designator (the last two digits of the year). Thus, December 15, 1993 is designated as 12/15/93. A date may also be represented in an alphanumeric form MMM/DD/YY, where MMM is an alphabetic month designator (e.g., DEC for December), and DD and YY are the same as in the numerical form. December 15, 1993 is represented in this format as DEC/15/93.

Such approaches for the representation of dates have worked well since the advent of computer databases, which has occurred in the twentieth century. Dates may be sorted in chronological order using the numerical representations. However, with the turn of the century at January 1, 2000, the representation and utilization of dates becomes more complex. Using the numerical form above, December 15, 2000 is represented as 12/15/00. If a numerical sort is performed on 12/15/93 and 12/15/00, the later date 12/15/00 sorts as the first-occurring date, an incorrect result.

Sets of dates spanning the turn of the century and associated with past, current, and future activities are now stored in many databases. When stored in the conventional formats discussed above, those dates will not readily be used and numerically sorted in chronological order. They may be manually converted to a more usable form in the sense that programs may be written to perform conversions, manipulations, and sorting. However, these programs typically require additional data fields for storage, which may be objectionable in some circumstances.

There is a need for an improved approach to the representation and utilization of dates in databases, and for converting the existing dates in databases to a more usable form. The present invention fulfills this need, and further provides related advantages.

SUMMARY OF THE INVENTION

The present invention provides an approach to the representation and utilization of dates stored symbolically in databases. Existing symbolic date representations are converted to a more useful form of symbolic date representations without the addition of new data fields, and in a manner that is performed automatically by the computer and requires no user input. The approach of the invention permits direct numerical sorting of dates.

In accordance with the invention, a method of processing dates stored in a database comprises the steps of providing a database with dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the dates falling within a 10-decade period of time. A 10-decade window with a Y_AY_B value for the first year of the ten-decade window is selected, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database. A century designator C_1C_2 is determined for each date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B. Each date in the database is formatted with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2.

In the case of most practical interest, the 10-decade period of time spans the year 2000 and begins with a year in which the second digit (Y_B in Y_AY_B) is 0 (zero). For any 10-decade period including the year 2000, if the decade designator Y_1 of the date in the database is numerically less than the decade designator Y_A of the first decade of the 10-decade period of time, the century designator C_1C_2 is "20". If Y_1 is equal to or greater than Y_A, C_1C_2 is "19". Dates in databases spanning more than 10 decades are not handled by this approach, but it is not expected that

3

this limitation will be significant for most commercial and industrial databases.

This approach works particularly well if the dates are represented in the format $C_1C_2Y_1Y_2M_1M_2D_1D_2$. The date Dec. 15, 2000 is represented in this format as 20001215, for example. Dates represented in this format may be directly sorted numerically by fast sorting techniques, and thereafter stored back in the database.

The present invention thus provides an efficient approach to converting and utilizing symbolic date representations in databases, which allows automatic processing of dates ranging from before to after the year 2000. The large number of dates represented in some databases may thereby be readily processed and utilized. Other features and advantages of the present invention will be apparent from the following more detailed description of the preferred embodiment, taken in conjunction with the accompanying drawings, which illustrate, by way of example, the principles of the invention. The scope of the invention is not, however, limited to this preferred embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a schematic representation of a computer database with date information therein; and

Figure 2 is a block flow diagram of a preferred approach for practicing the approach of the invention.

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 schematically depicts a computer 20 having a read-only or random-access memory 22, a mass-storage device 23, and a central processing unit 24 therein. Stored in the memory 22 or on the mass-storage device 23 is a database 26. The database includes information in the form of symbolic representations of dates and associated information such as events occurring on the respective dates. In a conventional approach, the dates are stored in a format such as $M_1M_2/D_1D_2/Y_1Y_2$ format. M indicates month information, D day information, and Y year information,

f

with the subscript 1 or 2 indicating the first or second digit of the designator, respectively. December 15, 1993 is stored as 12/15/93 or 12-15-93, and December 15, 2000 is stored as 12/15/00 or 12-15-00, for example. If a numerical sort is performed on these dates, 12/15/00 will sort chronologically prior to 12/15/93.

Figure 2 illustrates the approach of the invention. The computer database 26 is provided, numeral 30, having symbolic representations of dates stored therein. In some cases, the dates will be represented as discussed in the preceding paragraph. In other cases, an alphanumeric designator is used. In that approach, each date is stored as $M_aM_bM_c/D_1D_2/Y_1Y_2$ format, where $M_aM_bM_c$ is an alphabetical symbol such as JAN for January, FEB for February, etc. In that case, the month designator $M_aM_bM_c$ is first converted to the numerical form M_1M_2 by converting JAN to "01", FEB to "02", etc.

A 10-decade window is selected, numeral 32. That is, it is necessary that all dates in the database will be within some period of 10 decades, or 100 years. This limitation poses little problem for most industrial and commercial databases. The window may be arbitrarily selected. For example, the decade could begin with the 1950's and end with the 2040's, or it could begin with the 1980's and end with the 2070's. The 10-decade window will normally include some decades from the prior century and some from the new century.

The first year of the 10-decade window is represented by Y_AY_B. In a commonly utilized application, Y_B is 0 (zero), although the invention is not limited to this case. That is, the 1950's first decade would be represented by Y_A0 of "50", and the 1980's first decade would be represented by Y_A0 of "80". For this case, a century designator C_1C_2 for a date is determined, numeral 34, by comparing the value of Y_1, the first digit of the year designator for the date, with Y_A, the first digit of the first decade of the 10-decade window. C_1C_2 is assigned a first value if Y_1 is less than Y_A and a second value if Y_1 is equal to or greater than Y_A.

In the case of most interest, the 10-decade window includes decades earlier than the year 2000 and decades later than the year 2000, and Y_B is zero C_1C_2 is assigned "20" if Y_1 is less than Y_A and is assigned "19" if Y_1 is equal to or greater than Y_A. In that case and for example, if Y_A is 5, meaning that the decade

5

beginning in 1950 was selected as the first decade of the 10-decade window, and if Y_1Y_2 is "43", the century designator C_1C_2 is "20", indicating that the year in question in the database is 2043. On the other hand, if Y_1Y_2 is "63", the century designator C_1C_2 is "19", indicating that the year in question in the database is 1963. This selection process is performed in a completely automated fashion by the computer, without human input other than to select the starting date of the 10-decade window.

The symbolic representations of the dates in the database are reformatted with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2, numeral 36 of Figure 2. In one case that produces particularly advantageous results for many operations, such as chronological date sorting, the date is represented in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$. For example, the date 12/15/93 (December 15, 1993) is represented as 19931215 and the date 12/15/00 (December 15, 2000) as 20001215. A straightforward numerical sort of date data fields expressed in this form produces an accurate chronological ordering.

Once the symbolic representations of the dates are reformatted according to the procedures set forth above, the date information may be sorted, numeral 38, or otherwise manipulated, numeral 40, together with the entries associated with the dates. Such manipulation may include handling of data associated with the dates, storing the dates and associated information back in the data base, or other processes.

The approach of the invention has been implemented in a computer program, a copy of which is attached as Exhibit A. This program converts dates both before and after the year 2000.

The present invention provides an effective technique for reformatting symbolic representations of date information that is rapid and automated, and yields new symbolic representations of date information that are particularly amenable to further processing. Although a particular embodiment of the invention has been described in detail for purposes of illustration, various modifications and enhancements may be made without departing from the spirit and scope of the invention. Accordingly, the invention is not to be limited except as by the appended claims.

6

CLAIMS

What is claimed is:

Sub A1

1. A method of processing symbolic representations of dates stored in a database, comprising the steps of

providing a database with symbolic representations of dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each symbolic representation of a date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B; and

reformatting the symbolic representation of the date in the database with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2.

2. The method of claim 1, wherein the 10-decade window includes the decade beginning in the year 2000.

3. The method of claim 2, wherein the step of determining includes the step of

determining the first value as 20 and the second value as 19.

4. The method of claim 1, including an additional step, after the step of reformatting, of

A

sorting the symbolic representations of dates ~~in the database~~.

5. The method of claim 1, wherein the step of reformatting includes the step of

7

reformatting each symbolic representation of a date into the format $C_1C_2Y_1Y_2M_1M_2D_1D_2$.

6. The method of claim 5, including an additional step, after the step of reformatting, of

sorting the symbolic representations of dates ~~in the database~~ using a numerical-order sort.

A

7. The method of claim 1, wherein the step of providing a database includes the step of

converting pre-existing date information having a different format into the format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator and Y_1Y_2 is the numerical year designator.

9 ~~10~~

A 10 ~~8~~. The method of claim ~~7~~, including the additional step, after the step of reformatting, of

manipulating information in the database having the reformatted date information therein.

4. ~~9~~. The method of claim 1, wherein the step of selecting includes the step of

selecting Y_AY_B such that Y_B is 0 (zero).

A 9. ~~10~~. The method of claim 1, including an additional step, after the step of reformatting, of

storing the ~~sorted~~ symbolic representation of dates and their associated information back into the database.

Sub A2

11. A ~~method of~~ processing dates in a database, comprising the steps of

providing a database with ~~dates stored~~ therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is ~~the~~ numerical day designator and

8

Y_1Y_2 is the numerical year designator, all of the dates falling within a 10-decade period of time which includes the decade beginning in the year 2000;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B; and

reformatting each date in the database in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$; and

sorting the dates in the database using a numerical-order sort.

12. The method of claim 11, wherein the step of providing a database includes the step of

converting pre-existing date information having a different format into the format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator and Y_1Y_2 is the numerical year designator.

A

15 ~~13~~. The method of claim 14 ~~15~~ ~~11~~, including the additional step, after the step of sorting, of

manipulating information in the database having the reformatted date therein.

13 ~~14~~. The method of claim 11, wherein the step of selecting includes the step of

selecting Y_AY_B such that Y_B is 0 (zero).

14. ~~15~~. The method of claim 11, including an additional step, after the step of sorting, of

storing the sorted dates and their associated information back into the database.

08/725574

DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

ABSTRACT OF THE DISCLOSURE

Dates stored in symbolic form in a database are reformatted to permit easy manipulation and sorting of date-related information. Each date in M_1M_2, D_1D_2, and Y_1Y_2 format is converted to C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2 format. To accomplish the conversion, a 10-decade window starting on Y_AY_B is defined that encompasses all dates in the database. The value of C_1C_2 is determined by the relative values of Y_1Y_2 and Y_AY_B. The reformatted date information is particularly useful when the reformatting is in $C_1C_2Y_1Y_2M_1M_2D_1D_2$ format, because sorting by date is accomplished using a pure numerical-value sort.

08725574

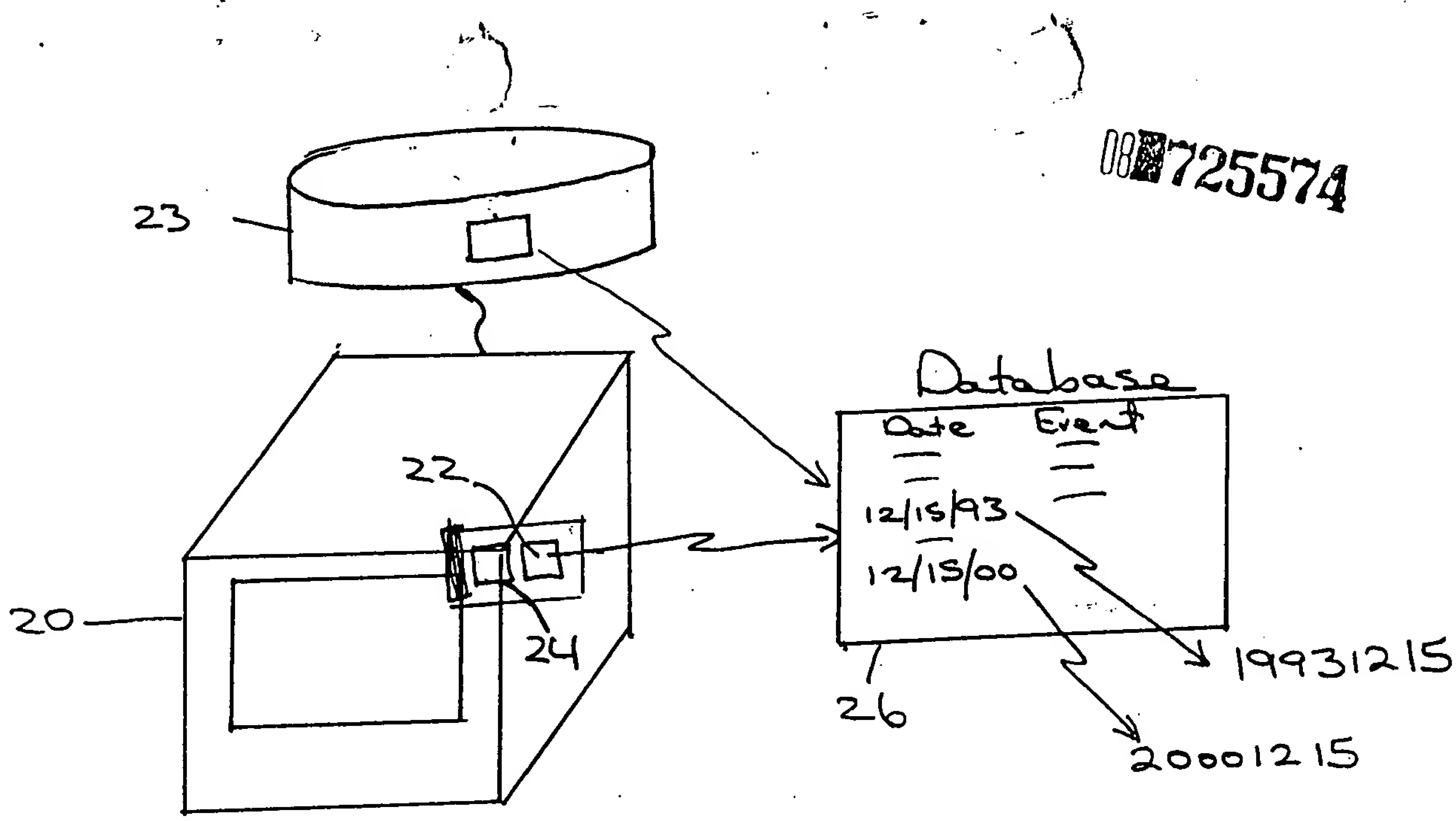


Fig 1

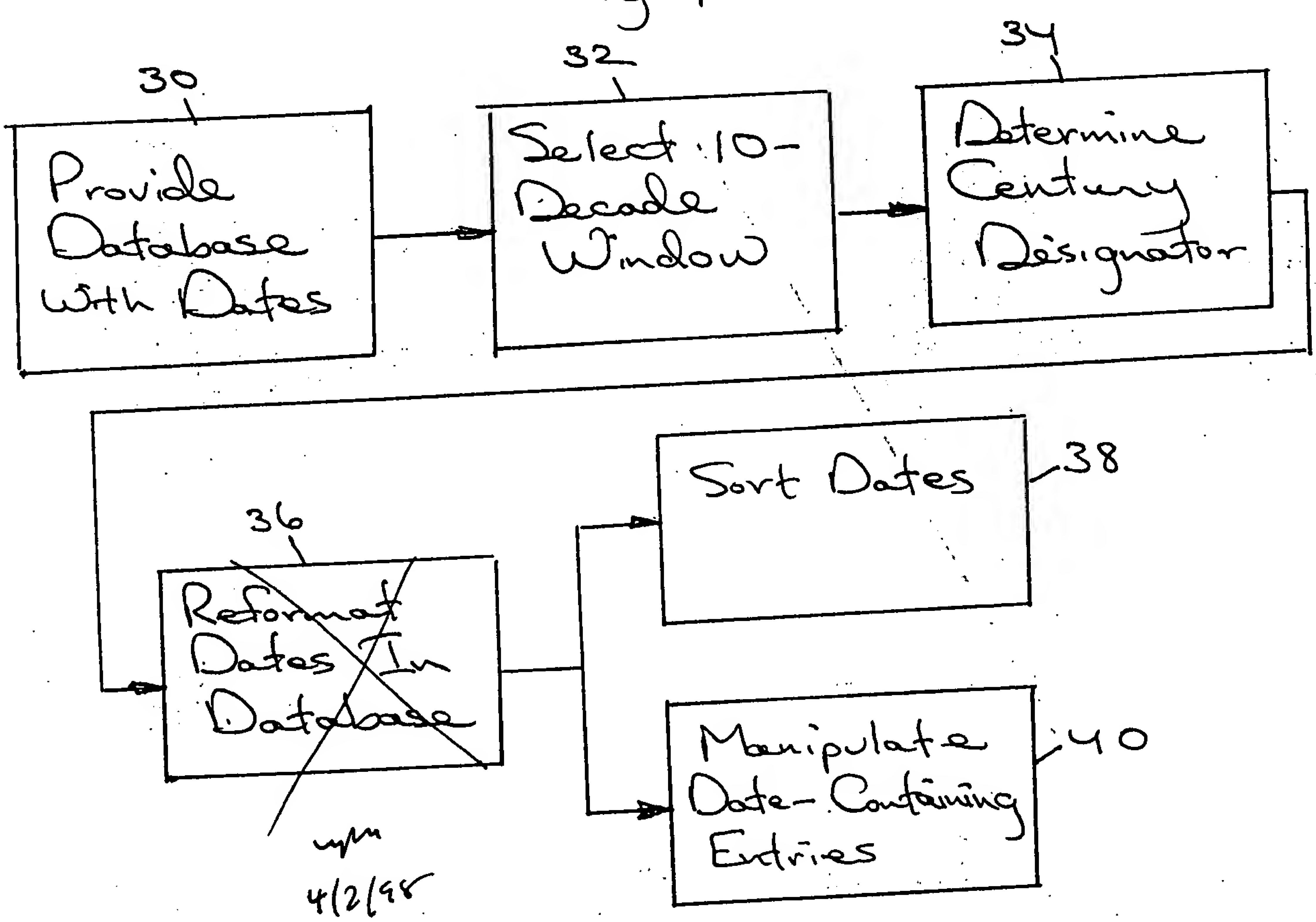


Fig2

725574

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

This declaration is of the following type:

_X_ original
____ design
____ supplemental
____ national stage of PCT
____ divisional
____ continuation
____ continuation-in-part (CIP)

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY, the specification of which (check one)

_X_ is attached hereto
_____ was filed on __________________
Application Serial No. ____________
and was amended on (or amended through _________
(if applicable)
____ was described and claimed in PCT international application No. _____ filed on _____ and as amended under PCT Article 19 on _______ (if any).

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Sec. 1.56.

____ In compliance with this duty there is attached an information disclosure statement. 37 CFR 1.97.

____ In compliance with this duty there has been filed an information disclosure statement on ________. 37 CFR 1.97.

I hereby claim foreign priority benefits under Title 35, United States Code, Sec. 119, of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

_X_ no such applications have been filed

___ such applications have been filed as follows:

Prior Foreign Application(s):

Number	Country	day/month/year filed	Yes No
None			

I hereby claim the benefit under Title 35, United States Code, Sec. 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Sec. 112, I acknowledge the duty to disclose all information known to be material to patentability as defined in Title 37, Code of Federal Regulations, Sec. 1.56(a) which became available between the filing date of the prior application and the national or PCT International filing date of this application.

Appln. Ser. No.	Filing Date	Status
None		

Power of Attorney: As a named inventor, I hereby appoint the following attorney(s) and/or agents to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith: RONALD L. TAYLOR, Registration No. 27,161, GREGORY O. GARMONG, Registration No. 29,382, and FELIX J. D'AMBROSIO, Registration No. 25,721.

Address all correspondence to:

Gregory O. Garmong
P.O. Box 12460
Zephyr Cove, NV 89448

Direct all telephone calls to GREGORY O. GARMONG at telephone No. (702) 588-0345. Direct any facsimile transmissions to GREGORY O. GARMONG at fax telephone No. (702) 588-0346.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of sole or first inventor: 1-00
BRUCE DICKENS

Inventor's signature: Bruce Dickens

Date: 9-6-, 1996

Residence: 3892 Cedron St., Irvine, CA 92714

Citizenship: United States

Post Office Address: 3892 Cedron St., Irvine, CA 92714

MAILROOM
JAN 31 1997
PAT. & TRADEMARK OFF.

Attorney's Docket No. 8190-119

PATENT

Gp 2209
#2
S. Bond
10/23/97

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Bruce Dickens
Serial No.: 08/725,574
Filed: October 3, 1996
For: DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

RECEIVED
97 FEB 24 PM 3:38
GROUP 240

RECEIVED
FEB 1 2 1997
GROUP 2300

Assistant Commissioner for Patents
Washington, DC 20231

REVOCATION OF POWER OF ATTORNEY
AND NEW POWER OF ATTORNEY BY ASSIGNEE

Sir:

Assignee hereby revokes all powers of attorney previously granted with respect to the above-identified patent application, and appoints:

Charles B. Elderkin
Reg. No. 24,357

Stephen M. Bodenheimer, Jr.
Reg. No. 28,932

Robert W. Glatz
Reg. No. 36,811

Guy R. Gosnell
Reg. No. 34,610

Jason P. Cooper
Reg. No. 38,114

of the firm of Bell, Seltzer, Park & Gibson and

Timothy H. Courson
Reg. No. 33,300

J. Rick Taché
Reg. No. 36,027

of McDonnell Douglas Corporation

as its attorney, with full power of substitution and revocation to transact all business in the Patent and Trademark Office in connection therewith.

Please direct all communications as follows:

Guy R. Gosnell
Registration No. 34,610
Bell, Seltzer, Park & Gibson, P.A.
Post Office Drawer 34009
Charlotte, NC 28234

Telephone (704) 331-6000
Facsimile (704) 334-2014

In re: Bruce Dickens
Serial No.: 08/725,574
Filed: October 3, 1996
Page 2

Assignee hereby elects under 37 C.F.R. § 3.71 to prosecute this patent application.

The undersigned Assignee hereby certifies that it is the assignee of the entire right, title, and interest in the patent application identified above by virtue of a chain of title from the inventor of the patent application identified above to the current assignee as shown below:

From Bruce Dickens to McDonnell Douglas Corporation, which Assignment was mailed with the original application on October 3, 1996 for recording in the Patent and Trademark Office.

The documents in the chain of title of the patent application identified above have been reviewed and, to the best of undersigned's knowledge and belief, title is in the assignee identified above.

The undersigned (whose title is supplied below) is empowered to sign this certificate on behalf of the Assignee.

I hereby declare that all statements made herein of my own knowledge are true, and that all statements made on information and belief are believed to be true; and further, that these statements are made with the knowledge that willful false statements, and the like so made, are punishable by fine or imprisonment, or both, under Section 1001, Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

MCDONNELL DOUGLAS CORPORATION

By: Richard J. Wickham
Richard J. Wickham

Title: Associate General Counsel

Date: 23 January 1997

257689
Atty. Dock. No. 8190-119

66199 U.S. PTO
04/16/97

Attorney Docket No. 08190-0119.000 PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

#3

In re: Dickens
Serial No.: 08/725,574
Filed: October 3, 1996
For: DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

Group Art Unit: 2309

Examiner:

RECEIVED
MAY -5 97
GROUP 2600

April 14, 1997

Assistant Commissioner for Patents
Washington, DC 20231

INFORMATION DISCLOSURE STATEMENT UNDER 37 C.F.R. § 1.97

Sir:

Attached is a list containing one document on form PTO-1449 together with a copy of the identified document. It is requested that this document be considered by the Examiner and officially made of record in accordance with the provisions of 37 C.F.R. § 1.97 and Section 609 of the MPEP.

Respectfully submitted,

Guy R. Gosnell
Registration No. 34,610

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, NC 28234
Telephone (704) 331-6000
Facsimile (704) 334-2014

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Assistant Commissioner for Patents, Washington, DC 20231, on April 14, 1997

Gwen Frickhoeffer
Date of Signature: April 14, 1997

269506

FORM PTO-1449 U.S. Department of Commerce Patent and Trademark Office	Attorney Docket Number 08190-0119.000
	Serial Number 08/72[illegible]
LIST OF DOCUMENTS CITED BY APPLICANT (Use several sheets if necessary)	Applicant: Dickens
	Filing Date: October 3, 1996 — Group 2309

661 99 U.S. PTO
04/16/97

RECEIVED
MAY -5 97
GROUP 2600

U. S. PATENT DOCUMENTS

Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date if Appropriate
[illegible]	A	4,573,127	02/1986	Korff	364	493	
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

FOREIGN PATENT DOCUMENTS

		Document Number	Date	Country	Class	Subclass	Translation Yes | No
	L						
	M						
	N						
	O						
	P						

OTHER DOCUMENTS (Including Author, Title, Date, Pertinent Pages, Etc.)

	Q	
	R	
	S	

EXAMINER Arnsbury	DATE CONSIDERED 11/6/97

*EXAMINER Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. 269505

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO.
08/725574	10/03/96	DICKENS	11151

EXAMINER

AMSBURY, WAYNE

ART UNIT | PAPER NUMBER
2307 | 4

GREGORY O GARMONG
PO BOX 12460
ZEPHYR COVE NV 89448

DATE MAILED: 10/23/97

This is in response to the Power of Attorney filed 01/31/97.

☐ 1. The Power of Attorney to you in this application **has been revoked** by the applicant. Future correspondence will be mailed to the new address of record. 37 CFR 1.33.

☑ 2. The Power of Attorney to you in this application **has been revoked** by the assignee who has intervened as provided by 37 CFR 3.71. Future correspondence will be mailed to the new address of record. (37 CFR 1.33).

☐ 3. The withdrawal as attorney in this application **has been accepted.** Future correspondence will be mailed to the new address of record. 37 CFR 1.33.

Susan Bond (Group 230)
This is a communication from the
Patent and Trademark Office

☑ 4. The Power of Attorney in this application **is accepted.** Correspondence in this application will be mailed to the below-noted address as provided by 37 CFR 1.33.

☐ 5. The Power of Attorney in this application **is not accepted** for the reason(s) checked below:

☐ a. The Power of Attorney is from an assignee and the Certificate required by 37 CFR 3.73 (b) has not been received.

☐ b. The person signing for the assignee has omitted their empowerment to sign on behalf of the assignee.

☐ c. The inventor(s) is without authority to appoint attorneys since the assignee has intervened as provided by 37 CFR 3.71.

☐ d. The signature of ____________________, a co-inventor in this application, has been omitted. The Power of Attorney will be entered upon receipt of confirmation signed by said co-inventor.

☐ e. The person(s) appointed in the Power of Attorney is not registered to practice before the U.S. Patent & Trademark Office.

☐ f. The revocation is not signed by the applicant, the assignee of the entire interest, or <u>one</u> particular principal attorney having the authority to revoke.

GUY R. GOSNELL
BELL, SELTZER, PARK & GIBSON, P.A.
P.O. DRAWER 34009

Susan Bond (Group 230)
This is a communication from the
Patent and Trademark Office

Notice of References Cited

Application No.	Applicant(s)	
08/725,574	Dickens	
Examiner	**Group Art Unit**	
Wayne Amsbury	2771	Page 1 of 1

U.S. PATENT DOCUMENTS

	DOCUMENT NO.	DATE	NAME	CLASS	SUBCLASS
A	5,630,118	5/13/97	Shaughnessy	707	1
B	5,644,762	7/1/97	Soeder	707	6
C	5,668,989	9/16/97	Mao	707	101
D					
E					
F					
G					
H					
I					
J					
K					
L					
M					

FOREIGN PATENT DOCUMENTS

	DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUBCLASS
N						
O						
P						
Q						
R						
S						
T						

NON-PATENT DOCUMENTS

	DOCUMENT (Including Author, Title, Source, and Pertinent Pages)	DATE
U	The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation, Third Edition, May, 1996.	5/1/96
V		
W		
X		

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR		ATTORNEY DOCKET NO.
08/725,574	10/03/96	DICKENS	B	11151

24M1/1117

GUY R. GOSNELL
BELL, SELTZER, PARK & GIBSON, P.A.
P.O. DRAWER 34009
CHARLOTTE NC 28234

EXAMINER	
AMSBURY, W	

ART UNIT	PAPER NUMBER
2307	5

DATE MAILED: 11/17/97

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner of Patents and Trademarks

PTO-90C (Rev. 2/95)

1- File Copy

Office Action Summary

Application No.	Applicant(s)
08/725,574	Dickens
Examiner	**Group Art Unit**
Wayne Amsbury	2307

☒ Responsive to communication(s) filed on *Oct 23, 1997*.

☐ This action is **FINAL**.

☐ Since this application is in condition for allowance except for formal matters, **prosecution as to the merits is closed** in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11; 453 O.G. 213.

A shortened statutory period for response to this action is set to expire *3* month(s), or thirty days, whichever is longer, from the mailing date of this communication. Failure to respond within the period for response will cause the application to become abandoned. (35 U.S.C. § 133). Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

Disposition of Claims

☒ Claim(s) *1-15* is/are pending in the application.

Of the above, claim(s) ______ is/are withdrawn from consideration.

☐ Claim(s) ______ is/are allowed.

☒ Claim(s) *1-15* is/are rejected.

☐ Claim(s) ______ is/are objected to.

☐ Claims ______ are subject to restriction or election requirement.

Application Papers

☒ See the attached Notice of Draftsperson's Patent Drawing Review, PTO-948.

☐ The drawing(s) filed on ______ is/are objected to by the Examiner.

☐ The proposed drawing correction, filed on ______ is ☐approved ☐disapproved.

☒ The specification is objected to by the Examiner.

☐ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. § 119

☐ Acknowledgement is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d).

☐ All ☐ Some* ☐ None of the CERTIFIED copies of the priority documents have been

☐ received.

☐ received in Application No. (Series Code/Serial Number) ______.

☐ received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

*Certified copies not received: ______.

☐ Acknowledgement is made of a claim for domestic priority under 35 U.S.C. § 119(e).

Attachment(s)

☒ Notice of References Cited, PTO-892

☒ Information Disclosure Statement(s), PTO-1449, Paper No(s). *3*

☐ Interview Summary, PTO-413

☒ Notice of Draftsperson's Patent Drawing Review, PTO-948

☐ Notice of Informal Patent Application, PTO-152

— SEE OFFICE ACTION ON THE FOLLOWING PAGES —

Serial Number: 08/725,574

Art Unit: 2307

CLAIMS 1-15 ARE PENDING

1. The disclosure is objected to because of the following informalities:

The statement of the problem, at the bottom of page 1, which implies that requiring additional data fields for storage to solve the Y2K problem, is contradicted by the use of century digits C_1C_2, as spelled out at page 2, second paragraph of the SUMMARY. Similarly, so is the second sentence of the SUMMARY.

Appropriate correction is required.

2. Claims 1-15 are rejected under 35 U.S.C. 112, first paragraph, as based on a disclosure which is not enabling. The "conversion of existing symbolic date representations ... without the addition of new data fields", as indicated at page 2 lines 7-10, is critical or essential to the practice of the invention, but not included in the claim(s) is not enabled by the disclosure. *In re Mayhew*, 527 F.2d 1229, 188 USPQ 356 (CCPA 1976).

The problem set forth in the last four lines of page 1 and promised in the first paragraph of page 2, as well as in the lines quoted above, indicate that the invention solves the Y2K problem without introducing additional digits. The claims, the abstract, and the description of the invention in the SUMMARY clearly involve century digits C_1C_2, which increase the number of date digits from 6 to 8, thus using 4 digits to indicate the year. One of ordinary skill in the art would not know how to resolve this discrepancy.

Serial Number: 08/725,574

Art Unit: 2307

3. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless --
>
> (a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

Claims 1-15 are rejected under 35 U.S.C. 102(a) as being clearly anticipated by The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation, Third Edition, May, 1996, IBM Corp.

Remark: The claims are clearly anticipated by the windowing techniques, discussed in some detail beginning at page 4-3. Note in particular the use of 19 and 20 for century digits at page 4-3. Sorting is addressed at 7-23 under DFSORT.

4. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

5. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Wayne Amsbury whose telephone number is (703) 305-3828. The examiner can normally be reached on Monday-Thursday from 6:30 AM to 5:00 PM Eastern time.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas G. Black, can be reached on (703) 305-9707. The fax phone number for this Art Unit is (703) 305-9731.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-9600.

WAYNE AMSBURY
PRIMARY PATENT EXAMINER
GROUP 2300

November 6, 1997

US PATENT & TRADEMARK OFFICE
3 0402 00115 3669

1

Third Edition

IBM

The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation

GC28-1251-02

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page xv.

Third Edition, May 1996

This is a major revision of, and obsoletes, GC28-1251-01.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405
FAX (Other Countries):
Your International Access Code +1+914+432-9405

IBMLink (United States customers only): KGNVMC(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL
Internet e-mail: mhvrcfs@vnet.ibm.com
World Wide Web: http://www.s390.ibm.com/

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Limited rights to copy the present work are hereby granted by the copyright owner named below. Accordingly, there is hereby granted the right to make a limited number of additional copies solely for the internal convenience of the recipient; no copies may otherwise be made. In particular, no copies may be made, no derivative works may be created and no compilations of the subject work may be created for purposes of republication, for redistribution, for sale, for rental, for lease or for any profit motivated activity whatsoever including the use of this work in support of or in conjunction with any service or service offering.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

©Copyright International Business Machines Corporation 1995, 1996.

Chapter 4. Reformatting Year-Date Notation

This chapter provides a number of techniques that you can employ to correct improper date notation and use. Because some techniques are appropriate only to unique situations, this section also lists the advantages, disadvantages, and IBM recommendations for their use.

When selecting a proposed Year2000 solution, evaluate the following factors:

1. What is the external impact due to incompatible date format changes?

   That is, what other programs or what output will be affected and to what extent will those programs require change if this solution is implemented for this particular program?

2. How current are the program modules that reference the date formats externalized by the exposures?

   That is, are there any plans to either eliminate or replace this particular program or routine, the programs that input to it, or those that receive or use its output?

3. What functions will be impaired due to Year2000 exposures?

   That is, will any mission-critical function within your company be compromised due to not reworking or replacing a particular program?

Solutions and Techniques

As you identify Year2000 exposures by the approaches described in Chapter 3, "Identifying 2-Digit-Year Exposures" on page 3-1, your next step is to rework the current program and data exposures to make your applications Year2000-ready. You can apply the following solutions to remove potential Year2000 exposures. Each solution is presented with an example technique to change the potential exposure. These suggested techniques require both program and data changes. Several solutions and techniques and their associated pros and cons follow:

Solution #1: Conversion to Full 4-Digit-Year Format

This solution is a 4-digit solution that externalizes a 4-digit-year format.

This approach requires changes to both the data and the programs by **converting all references and/or uses** of 2-digit-year format (YY) to 4-digit-year format (YYYY). It also requires that you convert all software programs that reference or use the updated data simultaneously, or use a 'bridging' mechanism to perform the conversion between old and new data and programs. Refer to "Bridge Programs Help Stage Format Conversions" on page 4-10 for more details on bridge programs. (You can accomplish this program and data conversion in steps.) Otherwise, you will immediately encounter data integrity problems caused by the inconsistency of date/time data formats.

To ease your migration, you might consider ignoring any non-impact (cosmetic) data fields in the YY format. A cosmetic date is one, that if externalized, is only interpreted by humans. Such occurrences might include the date on an output separator page or a display-only date on a screen in a panel-driven application.

© Copyright IBM Corp. 1995, 1996

Note: Be careful when selecting those situations that you decide to ignore and call cosmetic only. Be certain that they will not cause any data integrity exposures or ambiguity or are not accessed by any other program. Such instances of non-problem YY formats appear in a report header that shows the printing date of the report. The date is meant for human understanding only, not computer program manipulation. Consider the potential for future change. For example:

- Today's reports might be written to a data set tomorrow
- Display-only dates today may prove useful as a collating value when archiving that output tomorrow to meet a new business or government standard.
- Even when viewed by a human, 2-digit dates can prove ambiguous if the data spans 100 years.

If you allow the end user to continue to input 2-digit dates for compatibility and ease of data entry, then the responsibility to translate that data into a full 4-digit date falls to you, the application or systems programmer. One possible solution is to apply a context-sensitive prompt to allow the user to select a century indicator. For example, allow all dates to be entered as 2-digit dates and automatically prefix those with the current century unless the date is a future date or historical date. What constitutes "future" or "historical" is your decision but could be any date other than today's current day, week, month, year, and so on. Using this scheme, a future date in context of a loan maturity date could be set to 20yy, or a historical date automatically forces the user to select a century from a 'choose a century' (...16, 17, 18) prompt list.

Pros and Cons

Pros

1. Can provide 4-digit-year format. It is considered to be the **only** complete, permanent, and obvious solution.
2. Provides increased security against potential inappropriate decisions today if you do not selectively ignore 'cosmetic-only' situations.
3. Can ease your migration if you selectively ignore 'cosmetic-only' situations.

Cons

1. Need to convert the year data from 2-digit format to 4-digit format in all cases.
2. Requires that you relocate adjacent fields in the date field layout, and usually requires that you increase record lengths.
3. Inherent future risk in initial assessment that determined a particularly situation can be ignored as 'cosmetic only'.
4. Increased DASD space usage required due to data field expansion of data (consider including not only active but also archive data) and duplicate DASD space during conversion.
5. Might experience a performance impact due to increased time in processing and date access.
6. Some programming languages allow integer dates that are offset from a base date to be stored in files, data bases or passed as parameters between programs. Such integer dates provided by COBOL intrinsic functions, Language Environment callable services,

the CICS FORMATTIME command DAYCOUNT option, and other similar functions must conform to the standard YYYYMMDD format. This standard eliminates potential ambiguous data and errors due to each integer-date system using a unique starting date. Therefore, the potential for mixing incompatible integer dates when passed outside a single source module is extremely high and must be avoided.

Solution #2: Windowing Techniques

This is a 2-digit solution that externalizes either 2-digit or 4-digit-year formats. This approach requires changes to your programs only; no data changes are required.

CAUTION:
These approaches can be applied only to dates within a maximum 100-year period at any one time. This solution is considered temporary because there is no guarantee that in the future, your applications will not expand to process dates that are more than 100 years apart. Therefore, this approach always carries with it a potential future exposure. (For example, humans are living longer. Therefore data bases that include birthdays (medical, civil, insurance, and so on) and the applications that access that data are already at risk with many dates spanning 100+ years.)

Two types of **windowing** techniques have been defined: the fixed window technique and the sliding (rolling) window technique.

Fixed Window Technique

The **fixed window** technique uses a static 100-year interval that generally crosses a century boundary. This technique determines the century of a 2-digit year by comparing the 2-digit year against a window of 100 years. The user specifies the number of years in the past and future relative to a specific year within the 100-year interval.

Consider this specific example: if the years of date-related data of your application fall in the range of 1 January 1960 to 31 December 2059, you can use a 2-digit year to distinguish dates prior to the year 2000 from the year 2000 and beyond. If using the current system year of 1995, the number of years in the past and future are specified as 35 and 64, respectively. Program logic determines the century based on the following data checking. If the 2-digit year representation of a specific year is xy then if:

- xy≥60, then it is a 20th century date (19xy)
- Otherwise (that is, xy≤59), it is a 21st century date (20xy).

If, for example, you need to maintain a window of 35 past years and 64 future years, such that next year, 1996, your application can successfully deal with dates in the range 1961 through 2060, you need to adjust this program checking every year. The inherent future risk when employing this technique is obvious, and when compared to the sliding window technique is far less desirable.

Pros and Cons

Pros

1. No need to expand the 2-digit-year data to a 4-digit format.
2. Can provide 4-digit-year format for data reference.
3. Can distinguish years from different centuries using only 2-digit-year format (provided the years being processed are in the range of 100 years at any one time).
4. Can be useful if the particular program is being phased out, and a temporary solution is appropriate.

Cons

1. Potential exposures exist when/if the function of the software application needs to process years beyond the range of 100 years.
2. Expect a performance impact in direct proportion to the quantity of date processing the particular application handles due to the overhead of 2- to 4-digit-year conversion.
3. All programs that use the fixed window technique may need to be manually updated on a yearly basis depending on how your date routine is packaged.
4. All programs that accept output from the fixed window technique must use the same assumptions (current date, past and future windows).
5. Retaining a 2-digit year representation does not provide collating sequence support. Nor does the use of a fixed window technique provide indexing sequence support when 2-digit years are used as index keys in indexed files. You will need to provide additional processing to obtain correct collating and indexing sequence output.

Sliding Window Technique

The **sliding window** technique uses a self-advancing 100-year interval that generally crosses a century boundary. This technique determines the century of a 2-digit year by comparing the 2-digit year against a window of 100 years. The user specifies the number of years in the past and future relative to the system year (generally the current year) that the system sets[1] and maintains. Your applications can access the date that the system sets and automatically advances. This is the main advantage of using a sliding window over the fixed window (where the window is immovable without manually revising the programs each year).

As appropriate to your application environment, you can maintain more than one window. For example, you could set one window to process historical dates, one for mortgage dates, one for birth dates, and so on; and the program adjusts the system date and past and future windows to meet the specific application's needs.

Consider this specific example. If the dates in your application fall into a range of 35 years in the past and 64 years into the future, based on the current year, 1995,

[1] IBM product, Language Environment, provides an option whereby you can set the system year to other than the current year. This flexibility then allows you to set a 100-year range you require; it need not even contain the current year. Refer to Chapter 7, "Tool Categories and Available Tools to Ease Year2000 Changes" on page 7-1.

your program can accept and accurately deal with dates of 1960 through 2059. Next year, 1996, the window advances and your application accurately deals with dates of 1961 through 2060.

Graphically, Figure 4-1 on page 4-5 illustrates this example using the current (1995) 100-year window and that same window when the current system date has progressed to the year 2024.

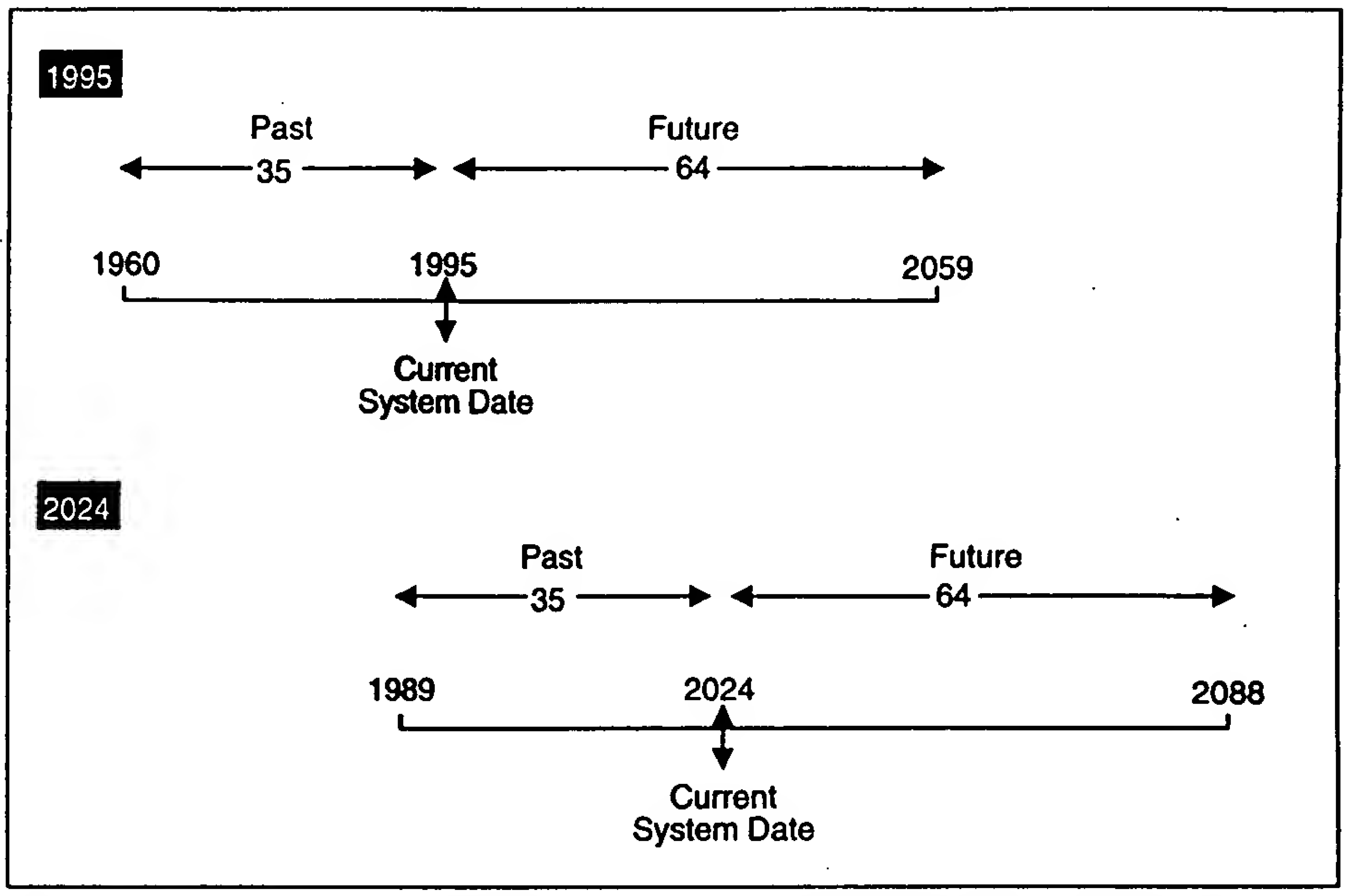


Figure 4-1. Graphical Representation of the Sliding Window Technique. (Using two current system dates, 1995 and 2024, as an example.)

A sliding window approach requires programming logic to interpret the meaning of all 2-digit year data. Such additional programming logic could be packaged into a common data/time service routine, callable from a 2-digit year data exploiter. This would reduce the programming overhead and impact to the calling programs. IBM product, Language Environment, provides common date/time service routines with sliding window features. By default, Language Environment uses a window of 80 years in the past and 20 years into the future that automatically adjusts based on the current year date. For details on using and setting the past/future window range, refer to Chapter 7, "Tool Categories and Available Tools to Ease Year2000 Changes" on page 7-1. For an example of how DFSORT/MVS is implementing a sliding window to sort, merge, and transform 2-digit year data to 4-digit year data, refer to "DFSORT" on page 7-23

Pros and Cons

Pros

1. No need to expand the 2-digit-year format to a 4-digit format.
2. Can provide 4-digit-year format for data reference.
3. Can distinguish years from different centuries using only 2-digit-year format (provided the years being processed are in the range of 100 years at any one time).

WITT automatically records and plays back all keystroke and mouse movements and compares and identifies test inconsistencies between benchmark test cases and test cases run after enhancements and fixes are made to a program. WITT also allows scripting in 2/REXX to add program intelligence to the test cases. This makes WITT test cases easy to update, maintain, and reuse for future testing.

WITT/OS2 installs on an OS/2 desktop and allows the testing of MVS, VM, IMS, CICS, OS/400, and OS/2 PM and Text (ie, Micro Focus, CICS/OS2) applications. WITT/PM installs on an OS/2 desktop and allows the testing of only OS/2 PM and Text applications. X/WITT installs on an AIX desktop and allows the testing of AIX X Window applications, as well as remote client applications in X Windows environments on SUN, HP, APOLLO, and MVS. WITT/Windows installs on a Windows 3.1 desktop and allows the testing of Windows workstation applications.

SORT

DFSORT V1R13 will enhance its Year2000 capabilities by providing the ability to sort, merge, and transform 2-digit years according to a specified sliding or fixed century window. New Y2C, Y2Z, Y2P, and Y2D formats, in conjunction with a new Y2PAST installation and run-time option, allow you to handle 2-digit year data in the following ways:

- Set the appropriate century window for your applications. For example, set a century window of 1915-2014 or 1950-2049.
- Order 2-digit character, zoned decimal, packed decimal, or decimal year data, according to the century window, using DFSORT's SORT and MERGE control statements. For example, order 96 (representing 1996) before 00 (representing 2000) in ascending sequence, or order 00 before 96 in descending sequence.
- Transform 2-digit character, zoned decimal, packed decimal, or decimal year data to 4-digit character year data, according to the century window, using DFSORT's OUTFIL control statement. For example, transform 99 to 1999 and 04 to 2004.

These DFSORT enhancements allow you to continue to use 2-digit years for sorting and merging, and assist those situations when you want to change 2-digit-year data to 4-digit-year data.

Additional information about DFSORT/MVS and its year2000 enhancements is available on the World Wide Web at URL:

`http://www.storage.ibm.com/storage/software/sort/srtmhome.htm`

Sliding Century Window

A new installation and run-time option allows you to specify a sliding or fixed century window to be used with 2-digit years. `Y2PAST=s` specifies a **sliding** century window starting s years before the current year. For example, if the current year is 1996, Y2PAST=80 starts the century window at 1996 - 80 = 1916, providing a century window of 1916 through 2015. In 1997, this century window automatically slides to 1917 through 2016.

`Y2KPAST=f` specifies a **fixed** century window starting at f. For example, Y2KPAST=1950 starts the century window at 1950, providing a century window of 1950 through 2049. Thus, Y2PAST allows you to control how DFSORT interprets the 2-digit years 00-99 on a site-wide or application-specific basis.

As an example, both Y2KPAST=1915 and Y2PAST=81 used in 1996 give a century window of 1915 through 2014, and result in the following interpretation of 2-digit year formatted data by DFSORT:

yy	Interpreted as:
00	2000
14	2014
15	1915
61	1961
62	1962
99	1999

2-Digit Year Formats

New formats allow you to identify 2-digit character, zoned decimal, packed decimal and decimal year data for special DFSORT processing as follows: (yy represents 2-digit year data in the examples below.)

Format	Meaning
Y2C	identifies 2-digit, 2-byte character year data such as C'yy', C'mm/dd/yy', or C'yy.mm.dd'
Y2Z	identifies 2-digit, 2-byte zoned decimal year data such as Z'yy', Z'mmddyy', or Z'yymmdd'
Y2P	identifies 2-digit, 2-byte packed decimal year data such as P'yy', P'dddyy', or P'yymmdd'
Y2D	identifies 2-digit, 1-byte decimal year data such as X'yy' or P'yyddd'

Sorting and Merging 2-Digit Years

You can use the new Y2C, Y2Z, Y2P, and Y2D formats in DFSORT's SORT and MERGE statements to identify specific 2-digit year data to be ordered according to the century window.

A simple example of the control statements to sort a C'mm/dd/yy' field (assume the current year is 1996) follows:

```
* Set the century window to 1962 through 2061
  OPTION Y2PAST=34
* Sort C'mm/dd/yy' as C'yymmdd'
  SORT FIELDS=(7,2,Y2C,A, * sort yy using century window
              1,2,CH,A,   * sort mm
              4,2,CH,A)   * sort dd
```

These control statements provide the following sort results:

Input Data (CH)	Sorted Output Data (CH)
06/22/15	03/18/62
10/03/00	09/01/99
11/14/61	10/03/00
08/16/14	08/16/14

09/01/99	08/17/14
03/18/62	06/22/15
08/17/14	11/14/61

Transforming 2-Digit Years to 4-Digit Years

You can use the new Y2C, Y2Z, Y2P, and Y2D formats in the OUTREC operand of DFSORT's OUTFIL statement to identify 2-digit year data to be changed to 4-digit year data according to the century window.

A simple example of the control statements to transform a P'yyddd' field follows:

```
* Set the century window to 1970 through 2069
  OPTION COPY,Y2PAST=1970
* Change P'yyddd' to C'yyyy/ddd'
 OUTFIL FNAMES=Y4,
  OUTREC=(1,1,Y2D,     * change X'yy' to C'yyyy' using
*                      *   century window
         C'/',         * insert C'/'
         2,2,PD,M11)   * change P'ddd' to C'ddd'
```

This code provides the following transformation results:

Input Data (HEX)	Transformed Output Data (CH)
92012F	1992/012
70225C	1970/225
69153F	2069/153
00001F	2000/001
99321F	1999/321
12054C	2012/054

COMUDAS (COMmon Uithoorn DAte Services)

COMUDAS (program product # 5788-HBB) is a common date routine that has been developed to replace all existing date routines, used by the IBM Uithoorn Lab, The Netherlands. COMUDAS offers all necessary functions for validation, conversion, and calculation of dates in any format. A standard interface must be used to call the date routine's load module dynamically.

The package also offers the possibility to use separate functions by means of NCAL's, that can be linked statically. This might be useful when converting large files or databases with validated data into other formats (for example, when reformatting year-date notation in an application).

Functions are available for updating the Calendar Tables by means of the Online Facility, that also can be used to test the Date Routines, and to print calendars.

A CICS version, as well as an MVS version, of this package is available.

IBM

(2)

GC28-1251-00

The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation

RECEIVED
SEP 4 - 1997
DIRECTOR'S OFFICE
GROUP 2300

Co:

First Edition, October 1995

Limited rights to copy the present work are hereby granted by the copyright owner named below. Accordingly, there is hereby granted the right to make a limited number of additional copies solely for the internal convenience of the recipient; no copies may otherwise be made. In particular, no copies may be made, no derivative works may be created and no compilations of the subject work may be created for purposes of republication, for redistribution, for sale, for rental, for lease or for any profit motivated activity whatsoever including the use of this work in support of or in conjunction with any service or service offering.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

About This Book ix
Who Should Use This Book ix
How to Use This Book ix
Of General Interest to Everyone ix
Executive and Senior-Level Management x
IS Managers x
System Programmers x
Application Programmers x
Executive Summary xi
Does this Really Mean Me? xi
But I've Been Told... xi
What's My Role and What Can I Expect? xii
What Are IBM and the Solution Developers Doing to Assist Me? xii
So, What's the Bottom Line? xiii

Notices xv
Trademarks xv
IBM Trademarks xv
Lotus Trademarks xvi
Non-IBM Trademarks xvi

Chapter 1. The Year 2000 - A Transition 1-1
Year2000 Exposure Classification 1-2
Scope of Year2000 Transition 1-3

Chapter 2. Planning to Resolve Your Year2000 Exposures 2-1
Planning Considerations 2-2
Inventory Your Software Portfolio 2-6

Chapter 3. Identifying 2-Digit-Year Exposures 3-1
Locating References 3-1
Tracing References Back to Their Source 3-2
Determining the Impact of 2-Digit-Year Data Fields 3-2
Investigating How Other Software Entities Use the Data 3-3
Data Sharing 3-3

Chapter 4. Reformatting Year-Date Notation 4-1
Solutions and Techniques 4-1
Solution #1: Conversion to Full 4-Digit-Year Format 4-1
Solution #2: Windowing Techniques 4-2
Solution #3: A 2-Digit Encoding/Compression Scheme 4-6
Using a Common Date/Time Service Routine 4-8
Considerations When Selecting Solutions 4-8
Solution Applicability 4-9
Bridge Programs Help Stage Format Conversions 4-9
Other Programming Situations 4-10
Guidelines 4-11

Chapter 5. Testing Techniques for Year2000 Changes 5-1
Structural Testing Techniques 5-1
Operations Testing 5-1

Stress Testing 5-1
Recovery Testing 5-2
Functional Testing Techniques 5-2
Requirements Testing 5-2
Regression Testing 5-2
Error Handling Testing 5-3
Manual Support Testing 5-3
Intersystem Testing 5-3
Parallel Testing 5-3
How to Change Date and Time for Testing 5-4
Basic Testing Scenarios 5-6
Basic Scenarios to Test Your PC System Clock 5-7

Chapter 6. Migration Consideration for Year2000 Transition 6-1
Plan for Migration 6-1
Perform Migration 6-3

Chapter 7. Tool Categories and Available Tools to Ease Year2000 Changes 7-1
Tool Characteristics 7-1
Tool Categories 7-2
Impact Analysis 7-2
Project Management 7-3
Program Level Analysis 7-3
Code Editing and Restructuring 7-4
Code Generation 7-5
Automate Testing 7-5
IBM Tools for MVS 7-7
The IBM COBOL Family for MVS & VM 7-7
The IBM PL/I Family for MVS & VM 7-18
COMUDAS (COMmon Uithoorn DAte Services) 7-23
IBM Tools for VM 7-26
The IBM COBOL Family for MVS & VM 7-26
The IBM PL/I Family for MVS & VM 7-26
REXX/EXEC Migration Tool for VM/ESA 7-26
IBM Tools for VSE 7-28
The IBM COBOL Family for VSE 7-28
The IBM PL/I Family for VSE 7-33
IBM Tools for AS/400 7-36
Using OPM RPG/400 Date Support 7-43
Using System/36 Compatible Date Support 7-43
The IBM COBOL Family for AS/400 7-44
The IBM C Family for AS/400 7-45
Integrated Language Environment for OS/400 7-46
DB2/400 SQL 7-47
OS/400 CL 7-53
Application Dictionary Services/400 7-59
Application Development Manager/400 7-60
IBM Tools for Personal Computers 7-61
The IBM COBOL Family for the Workstations 7-61
The IBM PL/I Family for the Workstations 7-67
Solution Developer Tools 7-71
Solution Developer Contacts and Product Name 7-71

Chapter 8. IBM Consulting and Services 8-1
TRANSFORMATION 2000: IBM's Century Date Change Solutions 8-1

Chapter 4. Reformatting Year-Date Notation

This chapter provides a number of techniques that you can employ to correct improper date notation and use. Because some techniques are appropriate only to unique situations, this section also lists the advantages, disadvantages, and IBM recommendations for their use.

When selecting a proposed Year2000 solution, evaluate the following factors:

1. What is the external impact due to incompatible date format changes?

   That is, what other programs or what output will be affected and to what extent will those programs require change if this solution is implemented for this particular program?

2. How current are the program modules that reference the date formats externalized by the exposures?

   That is, are there any plans to either eliminate or replace this particular program or routine, the programs that input to it, or those that receive or use its output?

3. What functions will be impaired due to Year2000 exposures?

   That is, will any mission-critical function within your company be compromised due to not reworking or replacing a particular program?

Solutions and Techniques

As you identify Year2000 exposures by the approaches described in Chapter 3, "Identifying 2-Digit-Year Exposures" on page 3-1, your next step is to rework the current program and data exposures to make your applications Year2000-ready. You can apply the following solutions to remove potential Year2000 exposures. Each solution is presented with an example technique to change the potential exposure. These suggested techniques require both program and data changes. Several solutions and techniques and their associated pros and cons follow:

Solution #1: Conversion to Full 4-Digit-Year Format

This solution is a 4-digit solution that externalizes a 4-digit-year format.

This approach requires changes to both the data and the programs by **converting all references and/or uses** of 2-digit-year format (YY) to 4-digit-year format (YYYY). It also requires that you convert all software programs that reference or use the updated data simultaneously, or use a 'bridging' mechanism to perform the conversion between old and new data and programs. Refer to "Bridge Programs Help Stage Format Conversions" on page 4-9 for more details on bridge programs. (You can accomplish this program and data conversion in steps.) Otherwise, you will immediately encounter data integrity problems caused by the inconsistency of date/time data formats.

To ease your migration, you might consider ignoring any non-impact (cosmetic) data fields in the YY format. A cosmetic date is one, that if externalized, is only interpreted by humans. Such occurrences might include the date on an output separator page or a display-only date on a screen in a panel-driven application.

Note: Be careful when selecting those situations that you decide to ignore and call cosmetic only. Be certain that they will not cause any data integrity

TRANSFORMATION 2000 Solutions . . . 8-1
Assessment and Strategy . . . 8-1
Detailed Analysis and Planning . . . 8-1
Implementation . . . 8-2
Year 2000 Clean Management . . . 8-2
Summary . . . 8-2

Appendix A. IBM Year2000-Ready Key Program Products and Hardware . . A-1
MVS . . . A-2
TPF . . . A-5
VSE/ESA . . . A-6
VM . . . A-8
OS/400 . . . A-10
AIX . . . A-13
OS/2 . . . A-17
Lotus Products . . . A-19
Hardware . . . A-21
IBM Personal Computer (PCs) - Hardware Timer Setting . . . A-21
Desktop PC Systems . . . A-21
Commercial PC Desktop Systems . . . A-25
Mobile PC Systems . . . A-26
PC Servers . . . A-28

Appendix B. Bibliography . . . B-1
Non-IBM Publications . . . B-1
By Author . . . B-1
By Title . . . B-3
Electronic (Internet World-Wide Web) Documentation . . . B-5
IBM Publications . . . B-6
By Author . . . B-6
Standard Publications . . . B-6

Appendix C. Glossary . . . C-1

Index . . . X-1

exposures or ambiguity or are not accessed by any other program. Such instances of non-problem YY formats appear in a report header that shows the printing date of the report. The date is meant for human understanding only, not computer program manipulation. Consider the potential for future change. For example:

- Today's reports might be written to a data set tomorrow
- Display-only dates today may prove useful as a collating value when archiving that output tomorrow to meet a new business or government standard.
- Even when viewed by a numan, 2-digit dates can prove ambiguous if the data spans 100 years.

Pros and Cons

Pros

1. Can provide 4-digit-year format
2. Provides increased security against potential inappropriate decisions today if you do not selectively ignore 'cosmetic-only' situations
3. Can ease your migration if you selectively ignore 'cosmetic-only' situations

Cons

1. Need to convert the year data from 2-digit format to 4-digit format in all cases.
2. Inherent future risk in initial assessment that determined a particularly situation can be ignored as 'cosmetic only'
3. Increased DASD space usage required due to data field expansion of data (consider including not only active but also archive data) and duplicate DASD space during conversion.
4. Might experience a performance impact due to increased time in processing and date access

Solution #2: Windowing Techniques

This is a 2-digit solution that externalizes either 2-digit or 4-digit-year formats. This approach requires changes to your programs only; no data changes are required.

CAUTION:
These approaches can be applied only to dates within a maximum 100-year period at any one time. This solution is considered temporary because there is no guarantee that in the future, your applications will not expand to process dates that are more than 100 years apart. Therefore, this approach always carries with it a potential future exposure. (For example, humans are living longer. Therefore data bases that include birthdays (medical, civil, insurance, and so on) and the applications that access that data are already at risk with many dates spanning 100+ years.)

Two types of **windowing** techniques have been defined: the fixed window technique and the sliding (rolling) window technique.

Fixed Window Technique

The fixed window technique uses a static 100-year interval that generally crosses a century boundary. This technique determines the century of a 2-digit year by comparing the 2-digit year against a window of 100 years. The user specifies the number of years in the past and future relative to a specific year within the 100-year interval.

Consider this specific example: if the years of date-related data of your application fall in the range of 1 January 1960 to 31 December 2059, you can use a 2-digit year to distinguish dates prior to the year 2000 from the year 2000 and beyond. If using the current system year of 1995, the number of years in the past and future are specified as 35 and 64, respectively. Program logic determines the century based on the following data checking. If the 2-digit year representation of a specific year is xy then if:

- xy≥60, then it is a 20th century date (19xy)
- Otherwise (that is, xy≤59), it is a 21st century date (20xy).

If, for example, you need to maintain a window of 35 past years and 64 future years, such that next year, 1996, your application can successfully deal with dates in the range 1961 through 2060, you need to adjust this program checking every year. The inherent future risk when employing this technique is obvious, and when compared to the sliding window technique is far less desirable.

Pros and Cons

Pros

1. No need to expand the 2-digit-year data to a 4-digit format.
2. Can provide 4-digit-year format for data reference.
3. Can distinguish years from different centuries using only 2-digit-year format (provided the years being processed are in the range of 100 years at any one time).
4. Can be useful if the particular program is being phased out, and a temporary solution is appropriate.

Cons

1. Potential exposures exist when/if the function of the software application needs to process years beyond the range of 100 years.
2. Expect a performance impact in direct proportion to the quantity of date processing the particular application handles due to the overhead of 2- to 4-digit-year conversion.
3. All programs that use the fixed window technique may need to be manually updated on a yearly basis depending on how your date routine is packaged.
4. All programs that accept output from the fixed window technique must use the same assumptions (current date, past and future windows).
5. Retaining a 2-digit year representation does not provide collating sequence support. Nor does the use of a fixed window technique provide indexing sequence support when 2-digit years are used as index keys in indexed files. You will need to provide additional

processing to obtain correct collating and indexing sequence output.

Sliding Window Technique

The **sliding window** technique uses a self-advancing 100-year interval that generally crosses a century boundary. This technique determines the century of a 2-digit year by comparing the 2-digit year against a window of 100 years. The user specifies the number of years in the past and future relative to the system year (generally the current year) that the system sets[1] and maintains. Your applications can access the date that the system sets and automatically advances. This is the main advantage of using a sliding window over the fixed window (where the window is immovable without manually revising the programs each year).

As appropriate to your application environment, you can maintain more than one window. For example, you could set one window to process historical dates, one for mortgage dates, one for birth dates, and so on; and the program adjusts the system date and past and future windows to meet the specific application's needs.

Consider this specific example. If the dates in your application fall into a range of 35 years in the past and 64 years into the future, based on the current year, 1995, your program can accept and accurately deal with dates of 1960 through 2059. Next year, 1996, the window advances and your application accurately deals with dates of 1961 through 2060.

Graphically, Figure 4-1 on page 4-5 illustrates this example using the current (1995) 100-year window and that same window when the current system date has progressed to the year 2024.

1 IBM product, Language Environment, provides an option whereby you can set the system year to other than the current year. This flexibility then allows you to set a 100-year range you require; it need not even contain the current year. Refer to Chapter 7, "Tool Categories and Available Tools to Ease Year2000 Changes" on page 7-1.

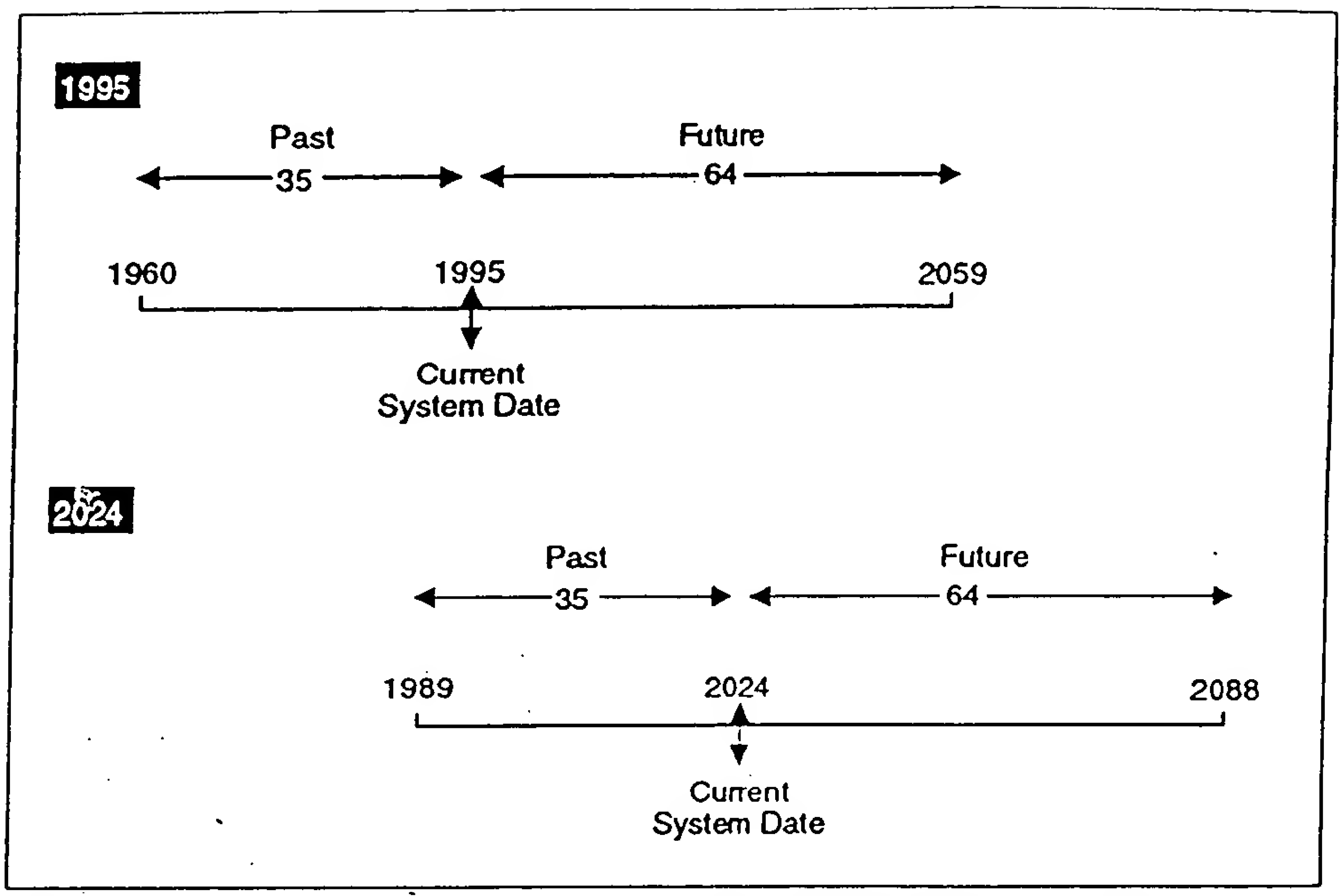


Figure 4-1. Graphical Representation of the Sliding Window Technique. (Using two current system dates, 1995 and 2024, as an example.)

A sliding window approach requires programming logic to interpret the meaning of all 2-digit year data. Such additional programming logic could be packaged into a common data/time service routine, callable from a 2-digit year data exploiter. This would reduce the programming overhead and impact to the calling programs. IBM product, Language Environment, provides common date/time service routines with sliding window features. By default, Language Environment uses a window of 80 years in the past and 20 years into the future that automatically adjusts based on the current year date. For details on using and setting the past/future window range, refer to Chapter 7, "Tool Categories and Available Tools to Ease Year2000 Changes" on page 7-1.

Pros and Cons

Pros

1. No need to expand the 2-digit-year format to a 4-digit format.
2. Can provide 4-digit-year format for data reference.
3. Can distinguish years from different centuries using only 2-digit-year format (provided the years being processed are in the range of 100 years at any one time).
4. No need to convert the date data to a new date representation scheme.

Cons

1. Potential exposures exist when/if the function of the software application needs to process years beyond the range of 100 years
2. Potential performance impact in direct proportion to the quantity of date processing the particular application handles
3. All programs that accept output from the sliding window technique must use the same assumptions (current date, past and future windows)

4. Retaining a 2-digit year representation does not provide collating sequence support. Nor does the use of a sliding window technique provide indexing sequence support when 2-digit years are used as index keys in indexed files. You will need to provide additional processing to obtain correct collating and indexing sequence output.

Solution #3: A 2-Digit Encoding/Compression Scheme

This is a 2-digit solution that externalizes only a 2-digit-year format. It requires changes to both your data and your programs. It also requires that you convert, simultaneously, all applications that reference or use the updated data.

Example techniques that are useful when using this solution include **encoding** or **compressing** 4-digit-year data into 2-digit existing space. This section presents several specific examples, many others exist and might prove more applicable to your specific needs.

CAUTION:
Apply this approach with caution. It is considered to be the least desirable approach and should only be used if absolutely necessary. Be certain that the new encoding or numbering scheme does not affect the proper functioning of your programs after all the data changes are implemented.

This solution is considered temporary because there is no guarantee that in the future, your applications will not expand to process dates that are outside the encoding limits.

Some examples include:

- **Example 1:** Convert the numbering scheme from decimal to hexadecimal. However two hexadecimal digits can provide only a maximum of 255 years, for example:

  ```
  D'1900' + X'FF' = 1900 + 255 = 2155
  ```

  Specific date conversions might be:

  - Convert the year 1900 represented by D'00' to X'00'
  - Convert the year 1999 represented by D'99' to X'63'
  - Convert the year 2000 represented by D'00' to X'64'

- **Example 2:** Convert the data type from the 2-byte character representation of the 2-digit year to a 1-byte **unsigned packed decimal** (two digit) representation, and use the freed byte to append two **unsigned packed decimal** digits to represent a 4-digit year. For example:

  - Convert the year 1900 (represented by character string '00' (= EBCDIC X'F0F0')) to unsigned packed decimal X'00' and prefix unsigned packed decimal X'19' in front of X'00' to yield X'1900' in unsigned packed decimal.
  - Convert the year 1999 (represented by character string '99' (= EBCDIC X'F9F9')) to unsigned packed decimal X'99' and prefix unsigned packed decimal X'19' in front of X'99' to yield X'1999' in unsigned packed decimal.
  - Convert the year 2000 (represented by character string '00' (= EBCDIC X'F0F0')) to unsigned packed decimal X'00' and prefix unsigned

packed decimal X'20' in front of X'00' to yield X'2000' in unsigned packed decimal.

- **Example 3:** Convert the numbering scheme from decimal to a user-defined numbering scheme. The mapping between the new and old schemes can be defined by a table or mapping function; and the conversion between the two numbering schemes can be done by table lookup or functional mapping. Figure 4-2 on page 4-7 presents one such possible user-defined table that provides for values up to 1295 within a 2-digit field. This scheme uses the characters 0-9 and A-Z to represent decimal values 0-35, respectively. This base 36 notation is thereby capable of extending the hexadecimal example on page 4-6 by 1040 more years. For example:

$$\text{D'1900'} + \text{base}_{36}\text{'ZZ'} = 1900 + 1295 = 3195$$

Figure 4-2. Example User-Defined Date/Year Conversion Table

2-Character Year (Encoded) Value	Converted Data/Year Value	Year (When Using 1900 as the Base Year)
00 - 0Z	00 - 35	1900 - 1935
10 - 1Z	36 - 71	1936 - 1971
20 - 2Z	72 - 107	1972 - 2007
30 - 3Z	108 - 143	2008 - 2043
40 - 4Z	144 - 179	2044 - 2079
⋮	⋮	⋮
R0 - RZ	972 - 1007	2872 - 2907
⋮	⋮	⋮
Z0 - ZZ	1260 - 1295	3160 - 3195

Pros and Cons

Pros

1. No need to expand the 2-digit-year data format to 4-digit data format. (For example, there is no need to increase the fields in data bases and tables to accommodate dates above 99 which would increase DASD usage.) Further, this saves the effort that would be required to rebuild your database(s).
2. Can distinguish years from different centuries using the 2-character-year format.

Cons

1. Depending upon the choice of data representation you implement, this scheme can be applied only to a limited date range. For example, you are limited to 255 years when using hexadecimal representation.
2. All programs that use this scheme and need to access the output of the 2-character conversion must change simultaneously.
3. Due to data conversion (calls and processing) you might experience a performance impact in direct proportion to the quantity of date processing the particular application handles.

4. Depending upon the choice of data representation you implement, you might experience incorrect data sequencing if you do not add further programming logic.
5. Encoded dates require conversion whenever you work with that data. Therefore, the presence of encoded dates will add another layer of complexity to such tasks as problem determination.

Using a Common Date/Time Service Routine

In large system applications it is common to find that more than one date/time service is in use. However, some date/time service routines may have Year2000 exposures of their own, for example, the routine(s) only provides a 2-digit-year format. Fixing the exposed date/time routine(s) is one possible solution. Selecting a vendor date/time routine that is certified as Year2000-ready for consolidation and/or replacement of your 'in-house' date/time service routines is another alternative.

While fixing your current date/time routine(s) exposures, you may find it worth your effort to consolidate all your date/time service routines into one **common date/time service routine**. If you then detect any Year2000 exposures during or following the consolidation, you can reformat your program and data and decide on the appropriate solution(s) you will use. That is, you can package any new code, encoding and conversion routines, windowing-specific data, and so on into the common date/time service routine. This common date/time service routine package might then be considered a 4-digit solution that externalizes both 2-digit and 4-digit-year formats. The benefit of using such a common date/time service routine is lower future maintenance because all services are consolidated rather than replicated throughout your applications.

Considerations When Selecting Solutions

As described in "Determining the Impact of 2-Digit-Year Data Fields" on page 3-2, potential 2-digit-year exposures can be classified into two categories (no impact and impact). When selecting an appropriate solution(s) for the 'impact' categories, be certain to consider not only the applicability of the solution(s) on the module itself but also the potential impact and adjustments on the external modules that receive data from this module. You have three basic choices; you can:

- Change your application
- Change your application and the data
- Invest in a new application (which could also require some date data changes)

Certainly, most IS organizations will build their 2000-ready system on a combination of these choices. When more than one solution appears feasible, weigh its appropriateness based on:

- Time available
- Resources available (personnel and hardware)
- Project cost (individual application conversions and overall)

As today's IS environment becomes increasingly more complex and sophisticated, the instances of program and data isolation decreases. Networking, open and distributed computing allow data to flow from site to site,

system program to application program (or application program to system program), and so on. You must ensure that these layers of software 'speak the same language'. As you add in-house code, Solution Developer-written code, and migrate your operating system, be certain to review that software for date format compatibility.

Solution Applicability

Different combinations of solutions are applicable to different situations. Evaluate solutions based on a 'best-solution combination' basis when considering both a module itself and other related modules. For example, when applying:

- **Solution #1** (full 4-digit solution) to a certain module, another module that receives data from this module could receive:
  - 2-digit-year data as before, provided there is no exposure for itself or
  - 2-digit-year data as before and apply Solution #2 (windowing techniques) for its own exposures or
  - 4-digit-year data and apply Solution #1 (full 4-digit solution) for its own exposure removal.
- **Solution #2** (windowing techniques) to a certain module, another module that receives data from this module may either receive 2-digit-year data as before and apply Solution #2 itself or receive 4-digit-year data and apply Solution #1 (full 4-digit solution) for its own exposure removal.
- **Solution #3** (the encoding or compression technique) to a certain module, another module that receives data from this module may need to apply Solution #3 as well to maintain data consistency with the data representation scheme. Another alternative is to apply Solution #3 to the impacted module, and then convert that 2-digit-year format to 4-digit-year format before externalizing that data to another module. This receiving module can then proceed with 4-digit data, and if necessary, apply Solution #1 (full 4-digit solution) to adopt the 4-digit-year data for its own exposure removal.

Bridge Programs Help Stage Format Conversions

Bridge programs are often used to convert data from one record format to another. If you use such a program, it should define the:

- Input date format and encoding method
- Output date format and encoding method
- Logic that converts the data from input format to output format based on their encoding methods.

You can apply bridge programs during program execution or file and/or database conversion. For application during program execution, the conversion occurs each time data is passed between programs or between program and source data using different record formats. For application during file and/or database conversion, the bridge program reads one record at a time from the source, transparently converts the record format, and writes out the data in the new format to the destination. The process is incremental and can continue until all the records in the source are converted.

Bridge programs for data format conversion provide the following benefits:

- Granularity when changing the code and/or data

With the scope of the Year2000 project, it is not practical (if possible) to change all the code and data at once. Bridge programs allow the gradual conversion of the programs and/or data and still maintain the compatibility between different data formats. For example, you can change some of your programs to adopt a new data format and still be able to communicate to programs using the old format (after conversion by the bridge programs). Therefore, changes to the remainder of your programs can be performed in an incremental manner as convenient.

- Flexibility when choosing appropriate solutions

  Bridge programs allow you to select the appropriate mix of different solutions to best meet your specific circumstances while maintaining the compatibility between different data formats. For example, you can design your programs so that they can process data in different formats. You can then have active data in a 4-digit-year format and archive the same type of data in 2-digit-year format. The bridge program distinguishes the data in these various formats by reading the records and, when necessary, converting the data to the appropriate format.

Other Programming Situations

Other programming situations you should consider might include:

- The possibility that a data format has become outdated and will not function correctly beyond 31 December 1999 (or earlier)

  Such data formats might be outdated even earlier and have already been superseded by another method by the Solution Developer. For example, IBM OS/VS Control Volumes (CVOLs) do not contain a mechanism to support dates beyond the end of 1999. In fact, if the current date plus the number of days that the data set is to be kept extends past 31 December 1999, the expiration date is interpreted to mean that the data set will be kept to the end of 1999. (On the MVS platform, IBM recommends that only ICF catalogs be used.)

- When migrating to year2000 support, your applications (operations) might support only 2-digit-year format, only 4-digit-year format, or both formats.

  It is possible that the 2-digit values are assumed to be 19xx dates. Therefore, be aware that all these must be eventually updated or the functions will fail or will give unpredictable results after 31 December 1999.

- Changes to operations procedures

  Be certain to educate your operators about command changes so that they know when they must use a full 4-digit date (for example, 2000, to avoid implying 1900 if they only enter 00).

- When erroneous data would be produced for a limited and known timeframe and changes are not justified

  You might have a situation that is best handled manually to meet a short time period where programming changes simply aren't justified. Consider using 2-digit-year data if a timeframe such as a single 24-hour period (31 December 1999 to 1 January 2000) or a single week (25 December 1999 through 1 January 2000) would be the only time your application will not provide correct results. For example, a program that looks at a sales report to compare the current day's merchandise movement with the previous 7 days. Because there would only be 8 reports containing both 1999 dates and

2000 dates, you might decide to handle the problem manually rather than changing the code.

Note: Don't fail to use a certain amount of common sense when deciding what applications to change, which to replace, and which to ignore. Do not lose your perspective of your institution's business needs and priorities and the impact and cost a particular application's change might have on attaining those goals.

Guidelines

While retaining a perspective of any external impact, module currency, and what functions are impaired due to Year2000 exposures, use the following guidelines when applying Year2000 solutions.

Note: This is not intended to be an exhaustive guideline, but rather a foundation upon which to start your specific Year2000 date-data resolution.

1. Establish an in-house 'date standard'. Conformance to the ISO Standard 8601 listed below would be a valuable starting point. The earlier such a standard is in place, the sooner your IS organization will avoid creating new date issues and the propagation of current ones. You can refer to:
   - ANSI X3.30-1985 (R1991) *Representation for Calendar Date and Ordinal Date for Information Interchange*
   - ANSI X3.51-1994 *Information Systems – Representations of Universal Time, Local Time Differentials, and United States Time Zone References for Information Interchange*
   - ISO 8601:1988 *Data elements and interchange formats – Information interchange – Representation of dates and times*

   These standards and ordering information are listed in the section entitled "By Title" in Appendix B, "Bibliography" on page B-1.
2. Minimize potential impact to external references due to incompatible date format changes. For example,
   - Maintain the 2-digit-year format as an option when 4-digit format is required for an application program interface (API) that provides 2-digit-year data references.
3. **Avoid** any ad-hoc solutions; such solutions inevitably require a future problem investigation and removal, and should be considered temporary solutions only. For example:
   - Do not determine the century of a 2-digit year by comparing the 2-digit year against a hard-coded threshold, for example, 60. If the 2-digit year is greater than or equal to 60, then the year is a 20th century year; otherwise, it is a 21st century year.
   - Do not fix the leap year calculation formula by adding logic to check if the current year is the year 2000. This solution will temporarily fix the leap year calculation problem by singling out the year 2000, but it does not fix the leap year calculation problem for other years in the multiple of 400.
4. A 2-digit-year format might be acceptable for human-only viewing purposes, for example, screen panels, hardcopy reports, and so on. However, any such data can be, and often is, added to a data set and then read by another program. A 'log' that can be used as input to any program should **not** be considered in this (non-impact, for human viewing only) category.

5. When changing the date format of any 'log', ensure that all the contributing programs adopt the new date format as well.

6. Consider the Year2000 solutions listed in this document (see "Solutions and Techniques" on page 4-1) for applicability in the following order.

   - Using a common date/time service routine (a 4-digit 'solution - that can support both 2- and 4-digit formats). This is:
     - Considered a long-term solution.
     - **The recommended solution** for its support of both 2- and 4-digit-year formats that provide a long-term solution and no impact to 2-digit-year data references.
   - Solution #1 (conversion to a full 4-digit-year format that externalizes 4-digit formats)
     - Considered a long-term solution.
     - Only supports 4-digit-year formats that will have impact on 2-digit year data reference.
   - Solution #2 (windowing techniques that externalize both 2- and 4-digit formats)
     - Considered a temporary solution and **should only be used when Solution #1 is not practical**. (This is an arguable issue, because there are applications that deal only with years in the range of 100 years. However, there is no guarantee that the functions of the applications will never change in the future and then require 4-digit-year formats.)
     - Has potential exposures when the function of the program needs to process years beyond the range of 100 years.
     - Use this solution only when:
       - Processing is always limited to the current date data, for example, at the time of IPL or time of job creation.
       - Expanding the date-data field is costly, and the function of the software program will be phased out before any exposure occurs.
   - Solution #3 (2-digit encoding/compression scheme that externalizes 2-digit formats)
     - Has potential exposures when the function of the program needs to process years beyond the range that can be covered by the encoding or compression scheme.
     - Should be used only when expanding the date-data field is costly and the function of the software program will be phased out before any exposure occurs.

Chapter 4. Reformatting Year-Date Notation

This chapter provides a number of techniques that you can employ to correct improper date notation and use. Because some techniques are appropriate only to unique situations, this section also lists the advantages, disadvantages, and IBM recommendations for their use.

When selecting a proposed Year2000 solution, evaluate the following factors:

1. What is the external impact due to incompatible date format changes?

   That is, what other programs or what output will be affected and to what extent will those programs require change if this solution is implemented for this particular program?

2. How current are the program modules that reference the date formats externalized by the exposures?

   That is, are there any plans to either eliminate or replace this particular program or routine, the programs that input to it, or those that receive or use its output?

3. What functions will be impaired due to Year2000 exposures?

   That is, will any mission-critical function within your company be compromised due to not reworking or replacing a particular program?

Solutions and Techniques

As you identify Year2000 exposures by the approaches described in Chapter 3, "Identifying 2-Digit-Year Exposures" on page 3-1, your next step is to rework the current program and data exposures to make your applications Year2000-ready. You can apply the following solutions to remove potential Year2000 exposures. Each solution is presented with an example technique to change the potential exposure. These suggested techniques require both program and data changes. Several solutions and techniques and their associated pros and cons follow:

Solution #1: Conversion to Full 4-Digit-Year Format

This solution is a 4-digit solution that externalizes a 4-digit-year format.

This approach requires changes to both the data and the programs by **converting all references and/or uses** of 2-digit-year format (YY) to 4-digit-year format (YYYY). It also requires that you convert all software programs that reference or use the updated data simultaneously, or use a 'bridging' mechanism to perform the conversion between old and new data and programs. Refer to "Bridge Programs Help Stage Format Conversions" on page 4-10 for more details on bridge programs. (You can accomplish this program and data conversion in steps.) Otherwise, you will immediately encounter data integrity problems caused by the inconsistency of date/time data formats.

To ease your migration, you might consider ignoring any non-impact (cosmetic) data fields in the YY format. A cosmetic date is one, that if externalized, is only interpreted by humans. Such occurrences might include the date on an output separator page or a display-only date on a screen in a panel-driven application.

© Copyright IBM Corp. 1995, 1996

Note: Be careful when selecting those situations that you decide to ignore and call cosmetic only. Be certain that they will not cause any data integrity exposures or ambiguity or are not accessed by any other program. Such instances of non-problem YY formats appear in a report header that shows the printing date of the report. The date is meant for human understanding only, not computer program manipulation. Consider the potential for future change. For example:

- Today's reports might be written to a data set tomorrow
- Display-only dates today may prove useful as a collating value when archiving that output tomorrow to meet a new business or government standard.
- Even when viewed by a human, 2-digit dates can prove ambiguous if the data spans 100 years.

If you allow the end user to continue to input 2-digit dates for compatibility and ease of data entry, then the responsibility to translate that data into a full 4-digit date falls to you, the application or systems programmer. One possible solution is to apply a context-sensitive prompt to allow the user to select a century indicator. For example, allow all dates to be entered as 2-digit dates and automatically prefix those with the current century unless the date is a future date or historical date. What constitutes "future" or "historical" is your decision but could be any date other than today's current day, week, month, year, and so on. Using this scheme, a future date in context of a loan maturity date could be set to 20yy, or a historical date automatically forces the user to select a century from a 'choose a century' (...16, 17, 18) prompt list.

Pros and Cons

Pros

1. Can provide 4-digit-year format. It is considered to be the **only** complete, permanent, and obvious solution.
2. Provides increased security against potential inappropriate decisions today if you do not selectively ignore 'cosmetic-only' situations.
3. Can ease your migration if you selectively ignore 'cosmetic-only' situations.

Cons

1. Need to convert the year data from 2-digit format to 4-digit format in all cases.
2. Requires that you relocate adjacent fields in the date field layout, and usually requires that you increase record lengths.
3. Inherent future risk in initial assessment that determined a particularly situation can be ignored as 'cosmetic only'.
4. Increased DASD space usage required due to data field expansion of data (consider including not only active but also archive data) and duplicate DASD space during conversion.
5. Might experience a performance impact due to increased time in processing and date access.
6. Some programming languages allow integer dates that are offset from a base date to be stored in files, data bases or passed as parameters between programs. Such integer dates provided by COBOL intrinsic functions, Language Environment callable services,

Pros and Cons

Pros

1. No need to expand the 2-digit-year data to a 4-digit format.
2. Can provide 4-digit-year format for data reference.
3. Can distinguish years from different centuries using only 2-digit-year format (provided the years being processed are in the range of 100 years at any one time).
4. Can be useful if the particular program is being phased out, and a temporary solution is appropriate.

Cons

1. Potential exposures exist when/if the function of the software application needs to process years beyond the range of 100 years.
2. Expect a performance impact in direct proportion to the quantity of date processing the particular application handles due to the overhead of 2- to 4-digit-year conversion.
3. All programs that use the fixed window technique may need to be manually updated on a yearly basis depending on how your date routine is packaged.
4. All programs that accept output from the fixed window technique must use the same assumptions (current date, past and future windows).
5. Retaining a 2-digit year representation does not provide collating sequence support. Nor does the use of a fixed window technique provide indexing sequence support when 2-digit years are used as index keys in indexed files. You will need to provide additional processing to obtain correct collating and indexing sequence output.

Sliding Window Technique

The **sliding window** technique uses a self-advancing 100-year interval that generally crosses a century boundary. This technique determines the century of a 2-digit year by comparing the 2-digit year against a window of 100 years. The user specifies the number of years in the past and future relative to the system year (generally the current year) that the system sets[1] and maintains. Your applications can access the date that the system sets and automatically advances. This is the main advantage of using a sliding window over the fixed window (where the window is immovable without manually revising the programs each year).

As appropriate to your application environment, you can maintain more than one window. For example, you could set one window to process historical dates, one for mortgage dates, one for birth dates, and so on; and the program adjusts the system date and past and future windows to meet the specific application's needs.

Consider this specific example. If the dates in your application fall into a range of 35 years in the past and 64 years into the future, based on the current year, 1995,

[1] IBM product, Language Environment, provides an option whereby you can set the system year to other than the current year. This flexibility then allows you to set a 100-year range you require; it need not even contain the current year. Refer to Chapter 7, "Tool Categories and Available Tools to Ease Year2000 Changes" on page 7-1.

the CICS FORMATTIME command DAYCOUNT option, and other similar functions must conform to the standard YYYYMMDD format. This standard eliminates potential ambiguous data and errors due to each integer-date system using a unique starting date. Therefore, the potential for mixing incompatible integer dates when passed outside a single source module is extremely high and must be avoided.

Solution #2: Windowing Techniques

This is a 2-digit solution that externalizes either 2-digit or 4-digit-year formats. This approach requires changes to your programs only; no data changes are required.

CAUTION:
These approaches can be applied only to dates within a maximum 100-year period at any one time. This solution is considered temporary because there is no guarantee that in the future, your applications will not expand to process dates that are more than 100 years apart. Therefore, this approach always carries with it a potential future exposure. (For example, humans are living longer. Therefore data bases that include birthdays (medical, civil, insurance, and so on) and the applications that access that data are already at risk with many dates spanning 100+ years.)

Two types of **windowing** techniques have been defined: the fixed window technique and the sliding (rolling) window technique.

Fixed Window Technique

The **fixed window** technique uses a static 100-year interval that generally crosses a century boundary. This technique determines the century of a 2-digit year by comparing the 2-digit year against a window of 100 years. The user specifies the number of years in the past and future relative to a specific year within the 100-year interval.

Consider this specific example: if the years of date-related data of your application fall in the range of 1 January 1960 to 31 December 2059, you can use a 2-digit year to distinguish dates prior to the year 2000 from the year 2000 and beyond. If using the current system year of 1995, the number of years in the past and future are specified as 35 and 64, respectively. Program logic determines the century based on the following data checking. If the 2-digit year representation of a specific year is xy then if:

- xy≥60, then it is a 20th century date (19xy)
- Otherwise (that is, xy≤59), it is a 21st century date (20xy).

If, for example, you need to maintain a window of 35 past years and 64 future years, such that next year, 1996, your application can successfully deal with dates in the range 1961 through 2060, you need to adjust this program checking every year. The inherent future risk when employing this technique is obvious, and when compared to the sliding window technique is far less desirable.

your program can accept and accurately deal with dates of 1960 through 2059. Next year, 1996, the window advances and your application accurately deals with dates of 1961 through 2060.

Graphically, Figure 4-1 on page 4-5 illustrates this example using the current (1995) 100-year window and that same window when the current system date has progressed to the year 2024.

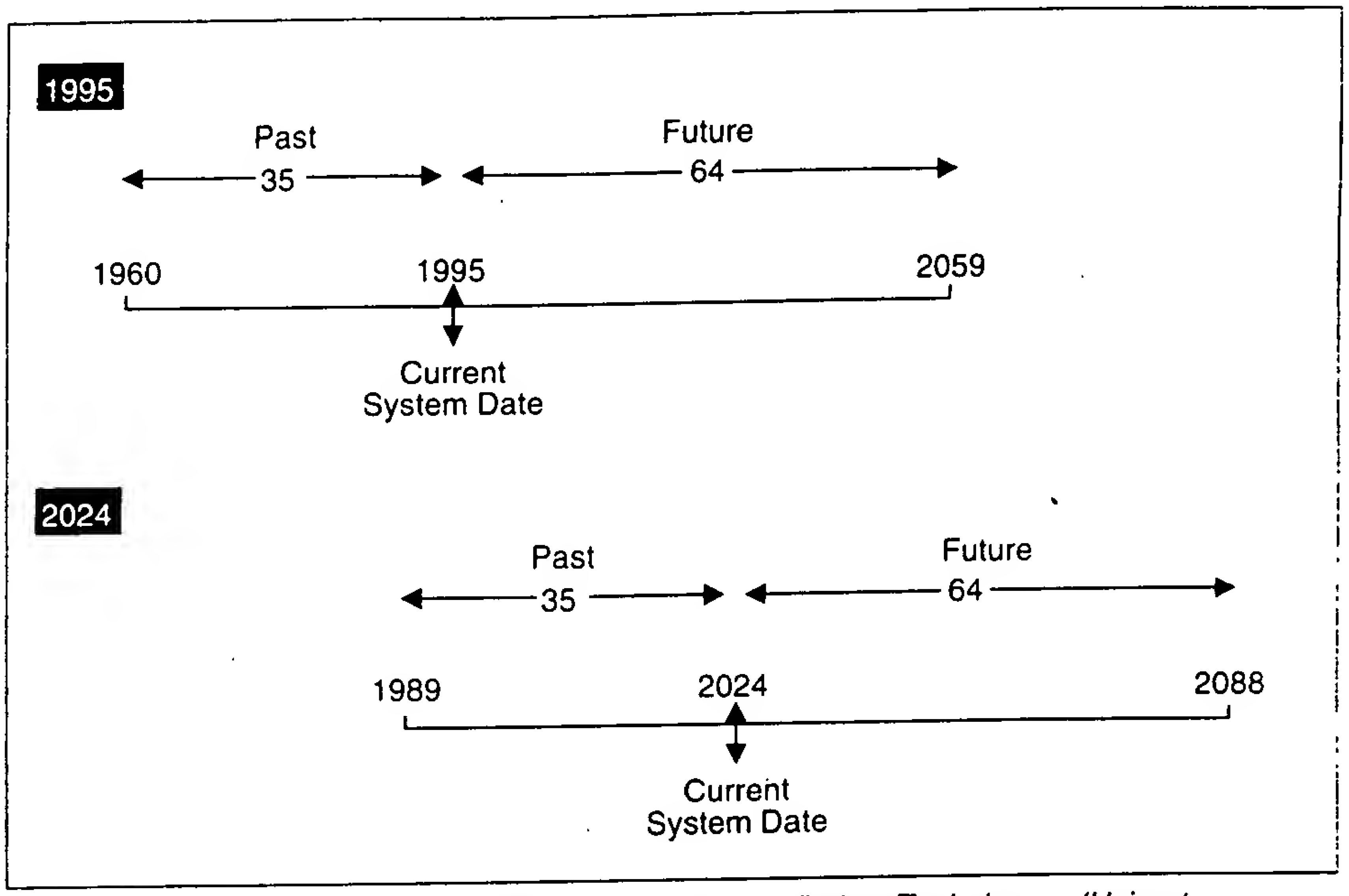


Figure 4-1. Graphical Representation of the Sliding Window Technique. (Using two current system dates. 1995 and 2024, as an example.)

A sliding window approach requires programming logic to interpret the meaning of all 2-digit year data. Such additional programming logic could be packaged into a common data/time service routine, callable from a 2-digit year data exploiter. This would reduce the programming overhead and impact to the calling programs. IBM product, Language Environment, provides common date/time service routines with sliding window features. By default, Language Environment uses a window of 80 years in the past and 20 years into the future that automatically adjusts based on the current year date. For details on using and setting the past/future window range, refer to Chapter 7, "Tool Categories and Available Tools to Ease Year2000 Changes" on page 7-1. For an example of how DFSORT/MVS is implementing a sliding window to sort, merge, and transform 2-digit year data to 4-digit year data, refer to "DFSORT" on page 7-23

Pros and Cons

Pros

1. No need to expand the 2-digit-year format to a 4-digit format.
2. Can provide 4-digit-year format for data reference.
3. Can distinguish years from different centuries using only 2-digit-year format (provided the years being processed are in the range of 100 years at any one time).

4. No need to convert the date data to a new date representation scheme.

Cons

1. Potential exposures exist when/if the function of the software application needs to process years beyond the range of 100 years
2. Potential performance impact in direct proportion to the quantity of date processing the particular application handles
3. All programs that accept output from the sliding window technique must use the same assumptions (current date, past and future windows)
4. Retaining a 2-digit year representation does not provide collating sequence support. Nor does the use of a sliding window technique provide indexing sequence support when 2-digit years are used as index keys in indexed files. You will need to provide additional processing to obtain correct collating and indexing sequence output.

Solution #3: A 2-Digit Encoding/Compression Scheme

This is a 2-digit solution that externalizes only a 2-digit-year format. It requires changes to both your data and your programs. It also requires that you convert, simultaneously, all applications that reference or use the updated data.

Example techniques that are useful when using this solution include **encoding** or **compressing** 4-digit-year data into 2-digit existing space. This section presents several specific examples, many others exist and might prove more applicable to your specific needs.

CAUTION:
Apply this approach with caution. It is considered to be the least desirable approach and should only be used if absolutely necessary. Be certain that the new encoding or numbering scheme does not affect the proper functioning of your programs after all the data changes are implemented.

This solution is considered temporary because there is no guarantee that in the future, your applications will not expand to process dates that are outside the encoding limits.

Some examples include:

- **Example 1:** Convert the numbering scheme from decimal to hexadecimal. However two hexadecimal digits can provide only a maximum of 255 years, for example:

  ```
  D'1900' + X'FF' = 1900 + 255 = 2155
  ```

  Specific date conversions might be:

  - Convert the year 1900 represented by D'00' to X'00'
  - Convert the year 1999 represented by D'99' to X'63'
  - Convert the year 2000 represented by D'00' to X'64'

- **Example 2:** Convert the data type from the 2-byte character representation of the 2-digit year to a 1-byte **unsigned packed decimal** (two digit) representation, and use the freed byte to append two **unsigned packed decimal** digits to represent a 4-digit year. For example:

- Convert the year 1900 (represented by character string '00' (= EBCDIC X'F0F0')) to unsigned packed decimal X'00' and prefix unsigned packed decimal X'19' in front of X'00' to yield X'1900' in unsigned packed decimal.
- Convert the year 1999 (represented by character string '99' (= EBCDIC X'F9F9')) to unsigned packed decimal X'99' and prefix unsigned packed decimal X'19' in front of X'99' to yield X'1999' in unsigned packed decimal.
- Convert the year 2000 (represented by character string '00' (= EBCDIC X'F0F0')) to unsigned packed decimal X'00' and prefix unsigned packed decimal X'20' in front of X'00' to yield X'2000' in unsigned packed decimal.

- **Example 3:** Convert the numbering scheme from decimal to a user-defined numbering scheme. The mapping between the new and old schemes can be defined by a table or mapping function; and the conversion between the two numbering schemes can be done by table lookup or functional mapping. Figure 4-2 on page 4-7 presents one such possible user-defined table that provides for values up to 1295 within a 2-digit field. This scheme uses the characters 0-9 and A-Z to represent decimal values 0-35, respectively. This base 36 notation is thereby capable of extending the hexadecimal example on page 4-6 by 1040 more years. For example:

  D'1900' + base$_{36}$'ZZ' = 1900 + 1295 = 3195

Figure 4-2. Example User-Defined Date/Year Conversion Table

2-Character Year (Encoded) Value	Converted Data/Year Value	Year (When Using 1900 as the Base Year)
00 - 0Z	00 - 35	1900 - 1935
10 - 1Z	36 - 71	1936 - 1971
20 - 2Z	72 - 107	1972 - 2007
30 - 3Z	108 - 143	2008 - 2043
40 - 4Z	144 - 179	2044 - 2079
⋮	⋮	⋮
R0 - RZ	972 - 1007	2872 - 2907
⋮	⋮	⋮
Z0 - ZZ	1260 - 1295	3160 - 3195

- **Example 4:** Pack a 3-digit date field with a 4-digit year date by the use of the CYY format. Using a conversion table or offset, you can indicate, for example, that C=0, 1, or 2 represents 19, 20, or 21, respectively. When your application appends the C to the YY field, your system produces full, 4-digit year dates, the range of which depends on the conversion mechanism. Using decimals 0-9 to represent 19-28, this scheme provides a solution from 1900 through 2899, but it is likely to require end-user procedural changes, education, and typical learning curve time and errors.

  One advantage to setting C=0 to represent 19 and so on is that it might provide a compatible, non-disruptive change to some existing application routines if such a field is currently prefixed to your YY data field and set to 0.

A variation on this same scheme would include the use of the CCYY format where the CC can be used to represent the actual century indicator, 18, 19, 20 and so on, or an encoded value for example, "00", "01", and "02", to represent 19, 20, 21, respectively (see "Solution #1: Conversion to Full 4-Digit-Year Format" on page 4-1).

When adding either the C or CC prefix to the YY field for CYY or CCYY representation, C or CC can be extracted from a separate field, one that is not necessarily adjacent to or preceding the YY field. This then can relieve any restrictions you might currently have due to your date field length. It does, however, require further programming logic and data manipulation.

Pros and Cons

Pros

1. No need to expand the 2-digit-year data format to 4-digit data format. (For example, there is no need to increase the fields in data bases and tables to accommodate dates above 99 which would increase DASD usage.) Further, this saves the effort that would be required to rebuild your database(s).
2. Can distinguish years from different centuries using the 2-character-year format.
3. If you use a COBOL COMP-3 format, you can pack a CCYYMMDD date into an existing 6-byte field (with one byte left over). This technique allows you to retain the original field size and eliminates your need to relocate adjacent fields. Applications that use the data for calculations run faster because the data is already packed.
4. If you use a flagged Julian format (CYYDDD) where C is used as the 'century indicator', the format does not require expansion of the date field.

Cons

1. Depending upon the choice of data representation you implement, this scheme can be applied only to a limited date range. For example, you are limited to 255 years when using hexadecimal representation.
2. All programs that use this scheme and need to access the output of the 2-character conversion must change simultaneously.
3. Due to data conversion (calls and processing) you might experience a performance impact in direct proportion to the quantity of date processing the particular application handles.
4. Depending upon the choice of data representation you implement, you might experience incorrect data sequencing if you do not add further programming logic.
5. Encoded dates require conversion whenever you work with that data. Therefore, the presence of encoded dates will add another layer of complexity to such tasks as problem determination.
6. You must convert the data before it can be displayed in Gregorian format, and some encoded data can only be viewed in hexadecimal format. This is both impractical for human reading and also impractical or impossible to print.

Using a Common Date/Time Service Routine

In large system applications it is common to find that more than one date/time service is in use. However, some date/time service routines may have Year2000 exposures of their own, for example, the routine(s) only provides a 2-digit-year format. Fixing the exposed date/time routine(s) is one possible solution. Selecting a vendor date/time routine that is certified as Year2000-ready for consolidation and/or replacement of your 'in-house' date/time service routines is another alternative.

While fixing your current date/time routine(s) exposures, you may find it worth your effort to consolidate all your date/time service routines into one **common date/time service routine**. If you then detect any Year2000 exposures during or following the consolidation, you can reformat your program and data and decide on the appropriate solution(s) you will use. That is, you can package any new code, encoding and conversion routines, windowing-specific data, and so on into the common date/time service routine. This common date/time service routine package might then be considered a 4-digit solution that externalizes both 2-digit and 4-digit-year formats. The benefit of using such a common date/time service routine is lower future maintenance because all services are consolidated rather than replicated throughout your applications.

Considerations When Selecting Solutions

As described in "Determining the Impact of 2-Digit-Year Data Fields" on page 3-3, potential 2-digit-year exposures can be classified into two categories (no impact and impact). When selecting an appropriate solution(s) for the 'impact' categories, be certain to consider not only the applicability of the solution(s) on the module itself but also the potential impact and adjustments on the external modules that receive data from this module. You have three basic choices; you can:

- Change your application
- Change your application and the data
- Invest in a new application (which could also require some date data changes)

Certainly, most IS organizations will build their 2000-ready system on a combination of these choices. When more than one solution appears feasible, weigh its appropriateness based on:

- Time available
- Resources available (personnel and hardware)
- Project cost (individual application conversions and overall)

As today's IS environment becomes increasingly more complex and sophisticated, the instances of program and data isolation decreases. Networking, open and distributed computing allow data to flow from site to site, system program to application program (or application program to system program), and so on. You must ensure that these layers of software 'speak the same language'. As you add in-house code, Solution Developer-written code, and migrate your operating system, be certain to review that software for date format compatibility.

Solution Applicability

Different combinations of solutions are applicable to different situations. Evaluate solutions based on a 'best-solution combination' basis when considering both a module itself and other related modules. For example, when applying:

- **Solution #1** (full 4-digit solution) to a certain module, another module that receives data from this module could receive:
  - 2-digit-year data as before, provided there is no exposure for itself or
  - 2-digit-year data as before and apply Solution #2 (windowing techniques) for its own exposures or
  - 4-digit-year data and apply Solution #1 (full 4-digit solution) for its own exposure removal.
- **Solution #2** (windowing techniques) to a certain module, another module that receives data from this module may either receive 2-digit-year data as before and apply Solution #2 itself or receive 4-digit-year data and apply Solution #1 (full 4-digit solution) for its own exposure removal.
- **Solution #3** (the encoding or compression technique) to a certain module, another module that receives data from this module may need to apply Solution #3 as well to maintain data consistency with the data representation scheme. Another alternative is to apply Solution #3 to the impacted module, and then convert that 2-digit-year format to 4-digit-year format before externalizing that data to another module. This receiving module can then proceed with 4-digit data, and if necessary, apply Solution #1 (full 4-digit solution) to adopt the 4-digit-year data for its own exposure removal.

Bridge Programs Help Stage Format Conversions

Bridge programs are often used to convert data from one record format to another. If you use such a program, it should define the:

- Input date format and encoding method
- Output date format and encoding method
- Logic that converts the data from input format to output format based on their encoding methods.

You can apply bridge programs during program execution or file and/or database conversion. For application during program execution, the conversion occurs each time data is passed between programs or between program and source data using different record formats. For application during file and/or database conversion, the bridge program reads one record at a time from the source, transparently converts the record format, and writes out the data in the new format to the destination. The process is incremental and can continue until all the records in the source are converted.

Bridge programs for data format conversion provide the following benefits:

- Granularity when changing the code and/or data

  With the scope of the Year2000 project, it is not practical (if possible) to change all the code and data at once. Bridge programs allow the gradual conversion of the programs and/or data and still maintain the compatibility between different data formats. For example, you can change some of your programs to adopt a new data format and still be able to communicate to programs using the old format (after conversion by the bridge programs). Therefore, changes

- Considered a long-term solution.
- **The recommended solution** for its support of both 2- and 4-digit-year formats that provide a long-term solution and no impact to 2-digit-year data references.

- Solution #1 (conversion to a full 4-digit-year format that externalizes 4-digit formats)
  - Considered a long-term solution.
  - Only supports 4-digit-year formats that will have impact on 2-digit year data reference.
- Solution #2 (windowing techniques that externalize both 2- and 4-digit formats)
  - Considered a temporary solution and **should only be used when Solution #1 is not practical**. (This is an arguable issue, because there are applications that deal only with years in the range of 100 years. However, there is no guarantee that the functions of the applications will never change in the future and then require 4-digit-year formats.)
  - Has potential exposures when the function of the program needs to process years beyond the range of 100 years.
  - Use this solution only when:
    - Processing is always limited to the current date data, for example, at the time of IPL or time of job creation.
    - Expanding the date-data field is costly, and the function of the software program will be phased out before any exposure occurs.
- Solution #3 (2-digit encoding/compression scheme that externalizes 2-digit formats)
  - Has potential exposures when the function of the program needs to process years beyond the range that can be covered by the encoding or compression scheme.
  - Should be used only when expanding the date-data field is costly and the function of the software program will be phased out before any exposure occurs.

to the remainder of your programs can be performed in an incremental manner as convenient.

- Flexibility when choosing appropriate solutions

  Bridge programs allow you to select the appropriate mix of different solutions to best meet your specific circumstances while maintaining the compatibility between different data formats. For example, you can design your programs so that they can process data in different formats. You can then have active data in a 4-digit-year format and archive the same type of data in 2-digit-year format. The bridge program distinguishes the data in these various formats by reading the records and, when necessary, converting the data to the appropriate format.

Other Programming Situations

Other programming situations you should consider might include:

- The possibility that a data format has become outdated and will not function correctly beyond 31 December 1999 (or earlier)

  Such data formats might be outdated even earlier and have already been superseded by another method by the Solution Developer. For example, The MVS platform will no longer support VSAM catalogs for processing when the system date is beyond 1999. To support data sets which need to have explicit expiration dates beyond the end of 1999, or to create cataloged data sets after 1999 on MVS systems, you must use ICF catalogs. Using VSAM catalogs on the MVS platform (including OS/390) will no longer function, this requires a programming change to take advantage of the alternative solution.

- When migrating to year2000 support, your applications (operations) might support only 2-digit-year format, only 4-digit-year format, or both formats.

  It is possible that the 2-digit values are assumed to be 19xx dates. Therefore, be aware that all these must be eventually updated or the functions will fail or will give unpredictable results after 31 December 1999.

- Changes to operations procedures

  Be certain to educate your operators about command changes so that they know when they must use a full 4-digit date (for example, 2000, to avoid implying 1900 if they only enter 00).

- When erroneous data would be produced for a limited and known timeframe and changes are not justified

  You might have a situation that is best handled manually to meet a short time period where programming changes simply aren't justified. Consider using 2-digit-year data if a timeframe such as a single 24-hour period (31 December 1999 to 1 January 2000) or a single week (25 December 1999 through 1 January 2000) would be the only time your application will not provide correct results. For example, a program that looks at a sales report to compare the current day's merchandise movement with the previous 7 days. Because there would only be 8 reports containing both 1999 dates and 2000 dates, you might decide to handle the problem manually rather than changing the code.

Note: Don't fail to use a certain amount of common sense when deciding what applications to change, which to replace, and which to ignore. Do not lose your perspective of your institution's business needs and priorities and the impact and cost a particular application's change might have on attaining those goals.

Guidelines

While retaining a perspective of any external impact, module currency, and what functions are impaired due to Year2000 exposures, use the following guidelines when applying Year2000 solutions.

Note: This is not intended to be an exhaustive guideline, but rather a foundation upon which to start your specific Year2000 date-data resolution.

1. Establish an in-house 'date standard'. Conformance to the ISO Standard 8601 listed below would be a valuable starting point. The earlier such a standard is in place, the sooner your IS organization will avoid creating new date issues and the propagation of current ones. You can refer to:

   - ANSI X3.30-1985 (R1991) *Representation for Calendar Date and Ordinal Date for Information Interchange*
   - ANSI X3.51-1994 *Information Systems -- Representations of Universal Time, Local Time Differentials, and United States Time Zone References for Information Interchange*
   - ISO 8601:1988 *Data elements and interchange formats -- Information interchange -- Representation of dates and times*

   These standards and ordering information are listed in the section entitled "By Title" in Appendix C, "Bibliography" on page C-1.

2. Minimize potential impact to external references due to incompatible date format changes. For example,

   - Maintain the 2-digit-year format as an option when 4-digit format is required for an application program interface (API) that provides 2-digit-year data references.

3. **Avoid** any ad-hoc solutions; such solutions inevitably require a future problem investigation and removal, and should be considered temporary solutions only. For example:

   - Do not determine the century of a 2-digit year by comparing the 2-digit year against a hard-coded threshold, for example, 60. If the 2-digit year is greater than or equal to 60, then the year is a 20th century year; otherwise, it is a 21st century year.
   - Do not fix the leap year calculation formula by adding logic to check if the current year is the year 2000. This solution will temporarily fix the leap year calculation problem by singling out the year 2000, but it does not fix the leap year calculation problem for other years in the multiple of 400.

4. A 2-digit-year format might be acceptable for human-only viewing purposes, for example, screen panels, hardcopy reports, and so on. However, any such data can be, and often is, added to a data set and then read by another program. A 'log' that can be used as input to any program should **not** be considered in this (non-impact, for human viewing only) category.

5. When changing the date format of any 'log', ensure that all the contributing programs adopt the new date format as well.

6. Consider the Year2000 solutions listed in this document (see "Solutions and Techniques" on page 4-1) for applicability in the following order.

   - Using a common date/time service routine (a 4-digit 'solution' - that can support both 2- and 4-digit formats). This is:

Form PTO 948 (Rev. 10-94) U.S. DEPARTMENT OF COMMERCE - Patent and Trademark Office Application No. 725574

NOTICE OF DRAFTSPERSON'S PATENT DRAWING REVIEW

PTO Draftpersons review all originally filed drawings regardless of whether they are designated as formal or informal. Additionally, patent Examiners will review the drawings for compliance with the regulations. Direct telephone inquiries concerning this review to the Drawing Review Branch, 703-305-8404.

The drawings filed (insert date) 10/3/96, are
A.____ not objected to by the Draftsperson under 37 CFR 1.84 or 1.152.
B. ✓ objected to by the Draftsperson under 37 CFR 1.84 or 1.152 as indicated below. The Examiner will require submission of new, corrected drawings when necessary. Corrected drawings must be submitted according to the instructions on the back of this Notice.

1. DRAWINGS. 37 CFR 1.84(a): Acceptable categories of drawings: Black ink. Color.
   - ___ Not black solid lines. Fig(s)____
   - ___ Color drawings are not acceptable until petition is granted. Fig(s)____
2. PHOTOGRAPHS. 37 CFR 1.84(b)
   - ___ Photographs are not acceptable until petition is granted. Fig(s)____
   - ___ Photographs not properly mounted (must use brystol board or photographic double-weight paper). Fig(s)____
   - ___ Poor quality (half-tone). Fig(s)____
3. GRAPHIC FORMS. 37 CFR 1.84 (d)
   - ___ Chemical or mathematical formula not labeled as separate figure. Fig(s)____
   - ___ Group of waveforms not presented as a single figure, using common vertical axis with time extending along horizontal axis. Fig(s)____
   - ___ Individuals waveform not identified with a separate letter designation adjacent to the vertical axis. Fig(s)____
4. TYPE OF PAPER. 37 CFR 1.84(c)
   - ___ Paper not flexible, strong, white, smooth, nonshiny, and durable. Sheet(s)____
   - ___ Erasures, alterations, overwritings, interlineations, cracks, creases, and folds copy machine marks not accepted. Fig(s)____
   - ___ Mylar, velum paper is not acceptable (too thin). Fig(s)____
5. SIZE OF PAPER. 37 CFR 1.84(f): Acceptable sizes:
   21.6 cm. by 35.6 cm. (8 1/2 by 14 inches)
   21.6 cm. by 33.1 cm. (8 1/2 by 13 inches)
   21.6 cm. by 27.9 cm. (8 1/2 by 11 inches)
   21.0 cm. by 29.7 cm. (DIN size A4)
   - ___ All drawing sheets not the same size. Sheet(s)____
   - ___ Drawing sheet not an acceptable size. Sheet(s)____
6. MARGINS. 37 CFR 1.84(g): Acceptable margins:

Paper size

	21.6 cm. X 35.6 cm. (8 1/2 X 14 inches)	21.6 cm X 33.1 cm. (8 1/2 X 13 inches)	21.6 cm. X 27.9 cm. (8 1/2 X 11 inches)	21.0 cm. X 29.7 cm. (DIN Size A4)
T	5.1 cm. (2")	2.5 cm. (1")	2.5 cm. (1")	2.5cm.
L	.64 cm. (1/4")	.64 cm. (1/4")	.64 cm. (1/4")	2.5 cm.
R	.64 cm. (1/4")	.64 cm. (1/4")	.64 cm. (1/4")	1.5 cm.
B	.64 cm. (1/4")	.64 cm. (1/4")	.64 cm. (1/4")	1.0 cm.

Margins do not conform to chart above. ✓
Sheet(s)____
✓ Top (T) ___ Left (L) ___ Right (R) ___ Bottom (B)

7. VIEWS. 37 CFR 1.84(h)
   REMINDER: Specification may require revision to correspond to drawing changes.
   - ___ All views not grouped together. Fig(s)____
   - ___ Views connected by projection lines or lead lines. Fig(s)____
   Partial views. 37 CFR 1.84(h) 2
   - ___ View and enlarged view not labled separately or properly. Fig(s)____
   Sectional views. 37 CFR 1.84 (h) 3
   - ___ Hatching not indicated for sectional portions of an object. Fig(s)____
   - ___ Cross section not drawn same as view with parts in cross section with regularly spaced parallel oblique strokes. Fig(s)____
8. ARRANGEMENT OF VIEWS. 37 CFR 1.84(i)
   - ___ Words do not appear on a horizontal, left-to-right fashion when page is either upright or turned so that the top becomes the right side, except for graphs. Fig(s)____
9. SCALE. 37 CFR 1.84(k)
   - ___ Scale not large enough to show mechanism with crowding when drawing is reduced in size to two-thirds in reproduction. Fig(s)____
   - ___ Indication such as "actual size" or scale 1/2" not permitted. Fig(s)____
10. CHARACTER OF LINES, NUMBERS, & LETTERS. 37 CFR 1.84(l)
   - ✓ Lines, numbers & letters not uniformly thick and well defined, clean, durable, and black (except for color drawings). Fig(s) 1-2
11. SHADING. 37 CFR 1.84(m)
   - ___ Solid black shading areas not permitted. Fig(s)____
   - ___ Shade lines, pale, rough and blurred. Fig(s)____
12. NUMBERS, LETTERS, & REFERENCE CHARACTERS. 37 CFR 1.84(p)
   - ✓ Numbers and reference characters not plain and legible. 37 CFR 1.84(p)(1) Fig(s) 1-2
   - ___ Numbers and reference characters not oriented in same direction as the view. 37 CFR 1.84(p)(1) Fig(s)____
   - ___ English alphabet not used. 37 CFR 1.84(p)(2) Fig(s)____
   - ✓ Numbers, letters, and reference characters do not measure at least .32 cm. (1/8 inch) in height. 37 CFR(p)(3). Fig(s) 1-2
13. LEAD LINES. 37 CFR 1.84(q)
   - ___ Lead lines cross each other. Fig(s)____
   - ___ Lead lines missing. Fig(s)____
14. NUMBERING OF SHEETS OF DRAWINGS. 37 CFR 1.84(t)
   - ___ Sheets not numbered consecutively, and in Arabic numerals, beginning with number 1. Sheet(s)____
15. NUMBER OF VIEWS. 37 CFR 1.84(u)
   - ___ Views not numbered consecutively, and in Arabic numerals, beginning with number 1. Fig(s)____
   - ___ View numbers not preceded by the abbreviation Fig. Fig(s)____
16. CORRECTIONS. 37 CFR 1.84(w)
   - ___ Corrections not made from prior PTO-948. Fig(s)____
17. DESIGN DRAWING. 37 CFR 1.152
   - ___ Surface shading shown not appropriate. Fig(s)____
   - ___ Solid black shading not used for color contrast. Fig(s)____

COMMENTS:

REMINDER

Drawing changes may also require changes in the specification, e.g., if Fig. 1 is changed to Fig. 1A, Fig. 1B, Fig. 1C, etc., the specification, at the Brief Description of the Drawings, must likewise be changed. Please make such changes by 37 CFR 1.312 Amendment at the time of submitting drawing changes.

INFORMATION ON HOW TO EFFECT DRAWING CHANGES

1. Correction of Informalities--37 CFR 1.85

File new drawings with the changes incorporated therein. The application number or the title of the invention, inventor's name, docket number (if any), and the name and telephone number of a person to call if the Office is unable to match the drawings to the proper application, should be placed on the back of each sheet of drawings in accordance with 37 CFR 1.84(c). Applicant may delay filing of the new drawings until receipt of the Notice of Allowability (PTOL-37). Extensions of time may be obtained under the provisions of 37 CFR1.136. The drawing should be filed as a separate paper with a transmittal letter addressed to the Drawing Review Branch.

2. Timing of Corrections

Applicant is required to submit **acceptable** corrected drawings within the three-month shortened statutory period set in the Notice of Allowability (PTOL-37). If a correction is determined to be unacceptable by the Office, applicant must arrange to have acceptable correction resubmitted within the original three-month period to avoid the necessity of obtaining as extension of time and paying the extension fee. Therefore, applicant should file corrected drawings as soon as possible.

Failure to take corrective action within set (or extended) period will result in **ABANDONMENT** of the Application.

3. Corrections other than Informalities Noted by the Drawing Review Branch on the Form PTO 948

All changes to the drawings, other than informalities noted by the Drawing Review Branch, **MUST** be approved by the examiner before the application will be allowed. No changes will be permitted to be made, other than correction of informalities, unless the examiner has approved the proposed changes.

OIPE JC21 MAR 20 1998 PATENT & TRADEMARK OFFICE

6 P. ~~2309~~ 2762
#6

Attorney's Docket No. 8190-119

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Dickens
Serial No.: 08/725,574 — Group No.: 2309
Filed: October 3, 1996 — Examiner: W. Amsbury
For: DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

March 17, 1998

Assistant Commissioner for Patents
Washington, DC 20231

PETITION AND FEE FOR EXTENSION OF TIME
(37 C.F.R. § 1.136(a))

1. This is a petition for an extension of time for a total period of 1 months to respond to the Official Action dated November 17, 1997.

2. A response in connection with the matter for which this extension is requested:

[X] is filed herewith.
[ ] has been filed.

RECEIVED
MAR 24 98
GROUP 2600

3. Applicant is
[ ] a small entity--verified statement:
[] attached.
[] already filed.
[X] other than a small entity.

4. Calculation of extension fee (37 C.F.R. § 1.17(a)(1)-(a)(5)):

	Total Months Requested	Fee For Other Than Small Entity	Fee for Small Entity
[X]	one month	$110.00	$55.00
[ ]	two months	$400.00	$200.00
[ ]	three months	$950.00	$475.00
[ ]	four months	$1,510.00	$755.00
[ ]	five months*	$2,060.00	$1,030.00

03/23/1998 SSMLEEKU 00000017 08725574
01 FC:115 110.00 OP

Fee Enclosed $ 110.00

Cannot be used to exceed six-month statutory limit for response to an Official Action.

[X] Charge Deposit Account No. 16-0605 for any additional extension and/or fee required or credit for any excess fee paid.

Respectfully submitted,

Guy R. Gosnell
Registration No. 34,610

BELL SELTZER INTELLECTUAL PROPERTY LAW GROUP
ALSTON & BIRD LLP
Post Office Drawer 34009
Charlotte, NC 28234
Tel (704) 331-6000
Fax (704) 334-2014
313946

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to:
Assistant Commissioner for Patents, Washington, DC 20231, on March 17, 1998.

Guy R. Gosnell

#7

OIPE
MAR 2 0 1998
JC21
PATENT & TRADEMARK OFFICE

Attorney's Docket No. 08190-0119.000 PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Bruce Dickens
Serial No.: 08/725,574
Filed: October 3, 1996
For: DATE FORMATTING AND SORTING
FOR SPANNING THE TURN OF
THE CENTURY

Group Art Unit: 2307
Examiner: W. Amsbury

March 17, 1998

Assistant Commissioner for Patents
Washington, DC 20231

RECEIVED
MAR 24 98
GROUP 2600

RESPONSE

Sir:

This Response is filed to address the issues raised by the Office Action dated November 17, 1997. As described below, Applicant submits that the disclosure is enabling and that both the disclosure and claims comply with 35 U.S.C. § 112, first paragraph. Further, the Applicant submits that the claimed invention was conceived and reduced to practice before the publication date of the cited IBM article. Thus, Applicant submits that the rejection of the claims under 35 U.S.C. § 102(a) is thereby overcome.

I. The Invention

As stated in the application, many existing databases contain date representations that are defined only by the decade and year designations (i.e., Y_1Y_2). Because these databases do not provide date designations for the century associated with each date, it will be impossible to discern the order of dates in a database after the turn of the century.

To properly and efficiently address this problem, a method for converting dates in databases was needed. This method should accept the dates from data storage, discern the

proper century designation for each date, and reformat the dates with the century designation.

The claimed invention provides a method of processing symbolic representations of dates stored in a database. The method of the claimed invention includes the step of providing a database that contains dates represented in symbolic representations in the form of $M_1M_2D_1D_2Y_1Y_2$, where M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator. The method of the claimed invention then selects a ten-decade window, beginning with the decade designated by Y_AY_B. Not only does Y_AY_B identify the first decade in the ten-decade period, but Y_AY_B is earlier than the earliest Y_1Y_2 year designator in the database (i.e., the earliest date in the database). The method of the claimed invention further includes the step of determining a century designator C_1C_2 for each symbolic representation of a date in the database based on the previously selected ten-decade period. In particular, the determining step will determine a first value for C_1C_2 if the Y_1Y_2 date representation for the date is less than Y_AY_B. On the other hand, the determining step will determine a second value for C_1C_2 if Y_1Y_2 is greater than Y_AY_B. Finally, the method of the claimed invention includes the step of reformatting the symbolic representations of the dates into corresponding values of C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2. These values can then be used to manipulate the dates, such as by sorting the dates in chronological order.

In a typical operation of the claimed invention, the method performs date conversions on a database that includes dates from both the twentieth and twenty-first century. In this operation, the method of the claimed invention initially selects a ten-decade window where the first year of the first decade is earlier than the earliest date in the database. For example, if the earliest date in the database is 1961, the

selecting step of the present method may choose a ten-decade period of 1960 - 2059.

After the selecting step has determined a proper ten-decade window, the method of the claimed invention determines a century designation C_1C_2 for each of the dates in the database. For example, if two date representations in the data base are 01/01/75 and 01/01/49, the determining step of the present invention would determine that 01/01/75 is 01/01/1975 because 75 > 60. Further, the determining step of the present invention would determine that 01/01/49 as 01/01/2049 because 49 < 60. The method of the present invention further includes the step of reformatting the symbolic representations of the dates into values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2. In other words, the date 01/01/75 is reformatted as 19570101 and the date 01/01/49 is reformatted as 20490101. These dates can then be used for several operations such as date sorting.

Advantageously, the method of the claimed invention can be implemented as an initial step in any database manipulation program. For instance, the method of the claimed invention may be embodied in computer software code that preprocesses a database prior to beginning the remainder of the data manipulation program. In this embodiment, the method initially converts the data from the varying formats, determines a century designation for each date, and reformats the dates such that the dates may be used by the database manipulation program for such operations as sorting and printing the dates. In this embodiment of the present invention, the dates are temporarily converted and reformatted for use by the manipulation program. However, the method of the present invention need not store the converted date in data storage. Instead, the original dates in data storage remain undisturbed. This aspect of the present invention thus allows conversion of dates to compensate for century

designations without requiring the addition of data fields to permanently store the century designations.

II. The Specification and Claims Comply with 35 U.S.C. § 112

In paragraph 1, the Office Action objects to the disclosure for implying that the current invention does not require additional data fields for storage to solve the year 2000 problem. Further, in paragraph 2, the Office Action rejected all of the claims under 35 U.S.C. § 112, first paragraph, as based on a disclosure that is not enabling. In particular, the Office Action states that the conversion of the existing symbolic date representations without the addition of new data fields is critical or essential to the invention but is not included in the claims.

As described below, the method of the claimed invention does not require that the converted data that includes the century designations be stored in data storage. Likewise, the Applicant respectfully submits that the amended set of claims does not require storage of the converted dates and therefore imposes no requirement for new data fields. Thus, Applicant respectfully submits that the disclosure is in accordance with 35 U.S.C. § 112 and is thus enabling.

As stated in the background of the invention, conventional date formatting systems typically require additional data fields for storage to accommodate the century designators. These additional data fields are necessary because conventional systems disclose a permanent reformatting of the stored data. The claimed invention, on the other hand, does not require that the reformatted data be permanently stored. Instead, the method of claimed invention encompasses embodiments in which the date information is initially reformatted and converted to have century designations, but does not require that the reformatted dates be stored. As stated previously, the method of one embodiment of the claimed

invention reads the dates from the database and temporarily reformats the dates with century designations. Data manipulation programs are then performed on these reformatted dates, such as sorting the dates. However, once the data manipulations are complete, the reformatted dates need not be stored in data storage. Instead, the dates in the data storage can remain the same as they were prior to the temporary reformatting of the data by the method of the claimed invention. Thus, in these embodiments, the method of the claimed invention does not require additional data fields for storage because the reformatted dates with century designations are only used "on the fly" for data manipulation and are not stored in data storage.

For the reasons described above, Applicant therefore submits that the disclosure is enabling and that both the disclosure and the claims comply with 35 U.S.C. § 112, first paragraph. As such, the rejections under 35 U.S.C. § 112 are therefore overcome.

III. The Claimed Invention was Reduced to Practice Prior to the Publication Date of the Cited Reference.

In paragraph 3, the Office Action rejected all of the claims under 35 U.S.C. § 102(a) as being anticipated by an IBM article entitled *The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation* (3d. ed. May 1996). Applicant submits herewith a Declaration under 37 C.F.R. § 1.131, signed by Bruce Dickens, the inventor of the above-identified application (hereinafter the "Dickens Declaration"). As stated in the Dickens Declaration, the method of the present invention was conceived and reduced to practice at least as early as April 4, 1996. (See Dickens Declaration, page 4, paragraph 9). In this regard, a computer program dated April 4, 1996 embodying one advantageous embodiment of the claimed invention was created at least as early as April 4, 1996.

This program was submitted as an Exhibit at the time of filing the present application and is attached as Exhibit G to the Dickens Declaration. As also set forth in paragraph 7 of the Dickens Declaration and as evident from the face of the computer program attached as an Exhibit to the present application and as Exhibit G to the Dickens Declaration, the April 4, 1996 date of the program predates the May 1996 publication date of the IBM article. Since the claimed invention was conceived and reduced to practice before the publication date of the cited IBM article, the third edition of the IBM article is therefore removed as a reference for purposes of 35 U.S.C. § 102(a). Thus, Applicant submits that the rejection of the claims under 35 U.S.C. § 102(a) is also thereby overcome.

In addition to the Declaration, the Applicant also submits herewith a Supplemental Information Disclosure Statement that discloses the first edition of the IBM article date October 1995. However, as stated in the Declaration, the Applicant conceived of the method of the claimed invention at least as early as February 1995. (See Dickens Declaration, page 2, paragraph 4). Further, the Applicant diligently worked to reduce to practice the claimed method from a date prior to October 1995 to its reduction to practice at least as early as April 4, 1996 as evidenced by the above-referenced computer program attached as Exhibit G to the Dickens Declaration. (See also Dickens Declaration, page 4, paragraph 9).

CONCLUSION

In view of the Declaration and the remarks presented above, it is respectively submitted that all of the present claims of the application are in condition for immediate allowance. It is therefore respectfully requested that a Notice of Allowance be issued. The Examiner is encouraged to

contact Applicant's undersigned attorney to resolve any remaining issues in order to expedite examination of the present application.

It is not believed that extensions of time are required, beyond those which may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any additional fee required therefore (including fees for net addition of claims) is hereby authorized to be charged to Deposit Account No. 16-0605.

Respectfully submitted,

Guy R. Gosnell
Registration No. 34,610

BELL SELTZER INTELLECTUAL PROPERTY LAW GROUP
ALSTON & BIRD LLP
Post Office Drawer 34009
Charlotte, NC 28234
Tel (704) 331-6000
Fax (704) 334-2014

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Assistant Commissioner of Patents, Washington, DC 20231, on March 17, 1998.

W. Kevin Ransom

314064

Exhibit A

OIPE JC21 MAR 20 1998 PATENT & TRADEMARK OFFICE

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Bruce Dickens
Serial No. 08/725,574
Filed: October 3, 1996
For: DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

Group Art Unit: 2771
Examiner: Wayne Amsbury

RECEIVED MAR 24 1998 GROUP 2800

Assistant Commissioner for Patents
Washington, DC 20231

DECLARATION UNDER 37 C.F.R. § 1.131

Sir:

I, Bruce Dickens, hereby declare and state that:

1. I am the sole inventor of the claimed invention of the above-identified U.S. Patent Application Serial No. 08/725,574 directed to a method of reformatting data stored in a database to overcome the problems arising from the improper recognition and processing of the century designations for dates.

2. I am presently employed by The Boeing Company in Irvine, California, which merged with my previous employer, McDonnell Douglas Corporation, on August 1, 1997. As such, I have worked continuously for first McDonnell Douglas Corporation and now The Boeing Company from February 6, 1988 until the present.

3. During the course of my employment with first McDonnell Douglas Corporation and now The Boeing Company, I have developed software to sort and otherwise process a variety of data. In order to properly process the data, I have had to develop software which reformats the data into substrings, each of which can be separately processed. As shown in Exhibit A, for example, I developed software that reformats a numerical representation of a tool into substrings for subsequent processing. As set forth at line 8 of the code attached at Exhibit A and designated by an "*", the character A represents the designation of different tools stored in a warehouse. For instance, the character A may represent the serial number and make of the part as "SN/Make." As such, the software that I developed and is attached as Exhibit A can reformat the character into two substrings (i.e., SN and Make) such that the tools can be separately sorted or otherwise processed by serial number and make based upon the substrings

"SN" and "Make", respectively. I developed the computer program attached as Exhibit A and named otmout.int;2 at least as early as September 27, 1994, as shown in the directory listing of Exhibit B. In this regard, Exhibit B is a printout of the directory including a listing for the computer program attached as Exhibit A and named "otmout.int;2" that indicates a creation date of September 27, 1994. (See Exhibit B, line 48, designated by an "*").

4. As early as February 1995, I realized that transforming data into substrings could be used to sort and otherwise process dates. For example, most dates are stored in databases as mm/dd/yy representations. I determined that these dates could be represented as substrings by removing the "/"s and reformatting the dates into substrings in the form of mmddyy. These substrings can then be used for sorting by month, day, or year. As such, I developed a computer program, fixmrr.int;2, which performs sorting of dates by reformatting dates into substrings. A printout of fixmrr.int;2 is attached at Exhibit C. As shown at page 5, line 5 of Exhibit C and also designated by an "**", the program reformats a date in the database, MRR(DATE), into a variable ADATE$ that is comprised of substrings. As also shown on page 5, line 11 of Exhibit C and designated by an "***", the program then sorts the dates by using the yy substring of the ADATE$ string. As shown in the directory printout attached as Exhibit D, the creation date of the fixmrr.int;2 program is at least as early as February 6, 1995. (See Exhibit D, line 11, designated by an "**").

5. As demonstrated by Exhibit E, I also developed other types of computer programs that reformatted dates into substrings. In this regard, Exhibit E is a printout of a computer program that determines the number of business days in a year. As can be seen from the output of this program set forth in Exhibit F, the working days of the year have been reformatted as 9998yyyymmdd. (See Exhibit F, col. 1). Further, in reference to Exhibit F, the dates of the year have been reformatted into substrings yymmddyynnn, wherein nnn represents the particular work day of the year. (See Exhibit F, col. 2). As indicated on page 1 of the computer program of

Exhibit E, this program was created at least as early as March 13, 1995.

6. During the development of the computer programs attached as Exhibits C and E and prior to October 1995, I conceived of the claimed method for date formatting and sorting of dates spanning the turn of the century based upon my determination that reformatting dates into substrings could be used to remedy the year 2000 problem. From a date prior to October 1995, I then worked diligently to reduce to practice the claimed invention of date formatting and sorting. For example, once the dates have been parsed into their respective month, day, and year designations, I determined that the yy substring of a date could be tested against the earliest year in a ten decade period to determine whether the date was prior to or after the year 2000. For instance, a date represented by ADATE$ could be reformatted as mmddyy. The yy substring could then be compared to the earliest year in a ten decade period to determine whether the date was in the twentieth or twenty-first century. If the year designation for a particular date in the database is less than this earliest year, the date is designated as corresponding to the next century. If the year designation is greater than this earliest year, however, the date is designated as corresponding to the current century. The year can then be properly designated as yyyy and the date can be reformatted as yyyymmdd for subsequent data sorting.

7. As evidenced by the computer program dated April 4, 1996 that is attached as Exhibit G, I eventually developed a computer program for date formatting and sorting for dates spanning the turn of the century as claimed in the above referenced patent application. (See Exhibit G, line 3, designated by an "*"). Thus, I reduced the claimed invention to practice at least as early as April 4, 1996.

8. I recently became aware of the first and third editions of an IBM article entitled "*The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation.*" The first edition of the article indicates a publication date of October 1995 and the third edition indicates a publication date of May 1996.

9. However, the above statements and information demonstrate that I conceived of a method for date formatting and sorting for dates spanning the turn of the century prior to October 1995. The above statements and attached exhibits also demonstrate that I diligently worked to reduce to practice the claimed method from a date prior to October 1995 to my reduction of practice of the claimed invention at least as early as April 4, 1996.

10. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application of any patent issued thereon.

Bruce Dickens
BRUCE DICKENS

Mar 17, 1998
DATE

313486

```
!   open #1:name 'list.dat', access output
!10 open structure TOOLS  ame "otms_src_dir:TOOLS"
20 extract structure PARTS
        SORT BY PARTS(MODEL)
     end extract
30 print  'PARTS info'
   for each PARTS
:       A = PARTS(UNIT_PRICE) *
!       AB = PARTS(UUNIT_PRICE)
!       AA = A[2:7]
!       B = DTYPE(A[2:7])
!       B = VAL(TOOLS(QTY))
!       B = VALID(TOOLS(QTY),NUMBER)
!       IF  B = 1  THEN
     print  PARTS(PARTNO),PARTS(UNIT_PRICE); ' ';PARTS(UUNIT_PRICE)
!       END IF
   next PARTS
   close structure PARTS
!   close #1
50 end
```

D55:[Users.Bruced Intouch.OTM] OTM.INT

Exhibit B

```
Directory D55:[USERS.BRUG[illegible].INTOUCH.OTM]

ADD_STR.INT;10                4   28-MAY-1994 10:57:04.38
ADD_STR2.INT;14               2   27-MAY-1994 12:20:27.46
ADD_STR_EX.INT;2              1   16-MAR-1995 13:45:20.70
ADD_STR_EX2.INT;7             2   18-MAR-1995 13:18:50.09
  D_STR_EX3.INT;8             2   18-MAR-1995 13:53:38.55
ADD_STR_PRT.INT;6             2   27-MAY-1994 13:49:11.49
ADD_STR_TOL.INT;5             2   27-MAY-1994 13:50:01.88
AR1.INT;4                     1   19-OCT-1994 07:24:24.49
AR2.INT;29                    2   19-OCT-1994 09:25:50.84
B.COM;3                       1    8-JUN-1994 13:23:18.98
CUSTOMER.DAT;2               15   30-MAY-1988 11:44:48.50
CUSTOMER.DEF;1               51    4-MAY-1988 11:34:50.58
CUSTOMER.FDL;3                3   30-JAN-1990 10:27:01.95
CUSTOMER.STR;1                1    4-MAY-1988 11:34:42.57
EXT_PAR_CHL.INT;1             2   19-AUG-1994 14:37:00.12
EXT_PAR_CHLD.INT;1            2   19-AUG-1994 14:20:57.50
EXT_PAR_CLD1.INT;1            2   20-AUG-1994 09:41:20.21
EXT_PRT.INT;1                 4    2-FEB-1995 12:48:06.68
GOV.DEF;1                    27   23-MAY-1994 13:26:47.33
GOV.FDL;2                     4   27-MAY-1994 12:08:01.97
GOV.INT;4                     2   25-MAY-1994 13:16:58.17
GOV.STR;1                     1   23-MAY-1994 13:25:57.88
GOVMOD.INT;5                 39   28-MAY-1994 10:58:36.03
GOVMODSUB.INT;5              44   28-MAY-1994 11:01:44.56
GOVRPT.INT;6                 38   28-MAY-1994 11:00:18.57
GOVSPMOD.INT;6               55   28-MAY-1994 11:03:10.78
GOVTPSUB.INT;6               43   28-MAY-1994 11:06:06.87
INVOICE.DAT;2                21   30-MAY-1988 11:46:15.32
INVOICE.DEF;2                51    4-MAY-1988 12:00:53.54
INVOICE.FDL;3                 3   30-JAN-1990 10:27:20.33
  VOICE.MAIN;2                5   19-OCT-1989 17:04:31.53
  VOICE.STR;1                 1    4-MAY-1988 12:00:27.50
LEDGER.LIS;1                  3   26-MAY-1994 12:06:25.90
LIST.DAT;4                  371   27-SEP-1994 07:27:50.38
MAINTAIN.INT;225            313   16-FEB-1993 11:36:00.31
MAINTAIN_DAV.INT;1          315   10-SEP-1992 11:16:26.20
MAINTAIN_HELP.INC;2
                             19   25-AUG-1989 16:15:31.14
MARGIN.INT;1                  1   25-MAY-1994 10:09:38.48
MSAF$RESOURCES.DIR;1
                              1   31-JAN-1995 15:13:15.62
NANCY1.GUIDE;1                1   16-AUG-1994 14:12:29.86
NANCY1.INT;10                33   18-AUG-1994 13:02:36.17
NONAME.INT;3                  1    5-NOV-1994 10:44:54.09
OTM.INT;36                    2   12-OCT-1996 09:58:09.60
OTMOUT.INT;2                  2   27-SEP-1994 07:27:38.18 *
PARTE_EXC1.RPT;10             1   27-MAY-1994 12:18:45.99
PARTE_EXC2.RPT;10             1   27-MAY-1994 12:18:46.38
PARTE_EXC3.RPT;10             1   27-MAY-1994 12:18:47.65
PRI_EX.INT;1                  1   24-MAY-1994 08:00:07.35
PRI_EX2.INT;12                2   25-MAY-1994 09:10:54.54
PRI_EX3.INT;6                 2   25-MAY-1994 09:59:37.23
PTY.CMD;6                     1   27-MAY-1994 12:15:58.83
PTY.INT;7                    14   27-MAY-1994 11:59:38.69
RR.DAT;1                    144   12-AUG-1994 09:54:35.18
RR.DEF;1                     27   18-JUL-1994 12:14:17.48
RR.FDL;2                      4   19-JUL-1994 06:34:53.54
  .MAIN;34                    5   13-AUG-1994 14:42:11.00
RR.STR;2                      1   18-MAR-1995 13:00:50.58
TOOL1.INT;34                 37   20-AUG-1994 13:39:38.83
TOOLS.DAT;2                 144    3-JUN-1993 12:20:39.12
TOOLS.DEF;1                  27    3-JUN-1993 10:08:30.19
TOOLS.FDL;3                   4    2-AUG-1994 10:55:23.28
```

Exhibit C

```
1       !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! Program:    MR[illegible].int Version 3.0-9
        ! Package:    In[illegible]ach Utilities
        ! Author:     BRUCE DICKENS
        ! Date:       October 1, 1992
        ! Purpose:    ON-LINE TOOL MATERIAL SYSTEM
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

10      set system : comment "Initializing report"
        ask system : mode mode$
        ask system, symbol 'otms_toolno$':value toolno$
!added one line above   !moved value of global symbol to toolno$
        show_stats = true
        if  mode$ = "BATCH"  or  mode$ = "OTHER"  then  show_stats = false
        string_dtype = 1
        declare dynamic field_data
        max_list = 50
        max_structure = 9
        max_titles% = 20
        dim title$(max_titles%)
        dim real total(max_list,0 to max_list)
        dim real extract_totals(max_list)
        dim break_hold$(0 to max_list)
        dim break_compare$(0 to max_list)
        dim integer page_break(0 to max_list)
        dim integer skip_break_totals(0 to max_list)
        dim integer level_count(0 to max_list)
        dim integer groupcount(0 to max_list)
        dim integer str_found(0 to max_structure)
        dim string break_desc$(0 to max_list)
        dim integer break_pos(0 to max_list)

20      !Special variable declarations go here
        today$ = date$(days(date$),3)
        cntrl_z$ = chr$(26)
        no_current_record = 7305
        out_ch = 2
        st_inx% = 0
        last_break_level% = 1
        no_stats = false
        ask margin old_margin
        ask system, logical 'guide_file_locator' : value guide_file_locator$
        if  guide_file_locator$ <> '' then  &
          open structure locator : name guide_file_locator$
        ! for those site that use a file locator system for placing datafiles

        USR_COUNT = 1

100     gosub main_logic
        end


1000    !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! M A I N   L O G I C
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine main_logic

1020    gosub init
        gosub open_files
      do
        do
          gosub extract_init
          if bailout% then exit do
          gosub extract_records
          if bailout% then exit do
          gosub print_init
```

```
    gosub print_report
    if bailout% then exit do
    message "Pages printed: " + str$(page_counter%)
  end do
  close #out_ch
  if bailout% then
    message error: 'Report aborted by user action...'
    delay 3
  else
    gosub display_report
  end if
  if bailout% then exit do
loop
  close all
  clear

1099    end routine


        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! I N I T
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! start up initialization
        !
        ! Expected:
        !
        ! Result   :
        !
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine init

        margin = 132
        set margin margin
        frame off
        gosub define_constants

        end routine


10000   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! D E F I N E   C O N S T A N T S
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine define_constants

10010   ! Special definitions go here
        tsuppress% = false
        making_report = true
        po_print_tag$ = "otm_"
        making_export = false
        making_export_structure = false
        unique_report_name = true
        report_name$ = "OTMS_RPT_DIR:OTM_MRR.LIS"
        report_margin = 255
        building_message$ = "Building report..."
        scr_width% = 132
        levels% = 0
        deepest_level% = 0
        totals_count% = 3
        do_totals% = true
        do_groupcount% = false
        titles% = 3
        lookup_err$ = "."
        spacing% = 1
        cutoff% = 0
```

```
          sample% = 0
          formsize$ = "66"
          lines_per_page% = 58
          break_hold$(0) = ""
          break_compare$(0) = ""
          page_break(0) = false
          skip_break_totals(0) = false
          break_desc$(0) = ""
          break_pos(0) =  1
          title$(1) = "On-Line Tool Material System"
!          title$(2) = "SPECIAL TOOLING CARD"
!          title$(3) =
          title_length% = 37
          report_width% = 126
          narrow% = false

10099     end routine


11000     !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          ! O P E N   F I L E S
          !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          routine open_files

11020     ! open structures here
          set system : comment "Opening data structures"
          if  guide_file_locator$ <> "" then
            set structure locator, field #1 : key "MRR"
            if  _extracted = 0 then
              message error : "Structure: MRR - Not found in " + guide_file_locatc
r$
              stop
            end if
            z$ = locator(#2)
            if  pos(z$, ":") = 0 then z$ = z$ + ":"
            open structure MRR: name "OTMS_SRC_DIR:MRR", datafile z$ + locator(#1)
          else
            open structure MRR: name "OTMS_SRC_DIR:MRR"
          end if

11099     end routine


          !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          ! E X T R A C T   I N I T
          !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          ! initialize variables needed for the extract phase
          !
          ! Expected:
          !
          ! Result  :
          !         rec        = 0
          !         sta        = 0
          !         found      = 0
          !         key_input$ = ''
          !         bailout%     = false
          !
          !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          routine extract_init

          rec% = 0
          sta% = 0
          found% = 0
          detail_rec% = 0
```

```
        page_counter% = 0
        key_input$ = ''
        model$ = ''
        bailout% = false


11200   u_prompt$  = "Tool Number? "
        u_default$ = ""
        u_len  = 20
        do
          gosub ask
          if  _exit  then
            bailout% = true
            exit do
          end if
          if  _back  then  repeat do

        tool_subset$ = ucase$(trim$(u_reply$))

        if tool_subset$ <> "" then toolno$ = tool_subset$
        if tool_subset$  = "" then tool_subset$ = toolno$
        set system, symbol 'otms_toolno$':value toolno$
!*added three lines above        !reset toolno$ to u_default$ and
                                 !set global symbol equal to toolno$
                                 !reset u_default$ to toolno$

        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        !  V A L D A T E   W O R K   O R D E R
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

!       set structure MRR, field toolno : partial key tool_subset$
        set structure MRR, field toolno :         key tool_subset$
          if _extracted = 0 then
            clear
             print at 10,35,bold,underline: "Tool subset not on File"
             delay
             repeat do

        end if
        end do

        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        gosub show_stats
        message "Extracting records..."
        set system : comment "Extracting from MRR"

        end routine


12000   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! E X T R A C T   R E C O R D S
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine extract_records

12020   !when exception in
        extract structure MRR
          gosub show_extract_status
          exclude ((cutoff% > 0) and (found% >= cutoff%))
          if  ((sample% > 0) and (found% >= sample%))  then  exit extract
          if bailout% then exit extract
          ! extract lookups go here
          ! LETS go here
          ! INCLUDES go here
```

```
    ! EXCLUDES go here
    ! SORTS go here
    ! EXTRACTION TOTALS go here

ADATE$ = MRR(DATE)[5:6] + MRR(DATE)[1:4] *
 include MRR (TOOLNO)[1:len(tool_subset$)] = tool_subset$
INCLUDE MRR ( TOOLNO )  =  tool_subset$

   when exception in
   sort  by MRR ( DATE )
   sort by ADATE$ **
    use
   end when
   if  _error  then  exclude true
   found% = found% + 1
   end extract

gosub show_recs_searched
gosub show_recs_selected
end routine


!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! P R I N T   I N I T
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! initialize variables needed during the print phase
!
! Expected:
!
! Result  :
!       The ouput file is opened
!        start_print_time = time(0)
!        first_line       = true
!        key_input$       = ''
!        page_counter%     = 0
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
routine print_init

message building_message$
set system : comment building_message$

if  out_ch > 0 and not making_export_structure then
  if  unique_report_name then
    open #out_ch: name report_name$, access output, unique
    ask #out_ch : name report_name$
  else
    open #out_ch: name report_name$, access output
  end if
  set #out_ch: margin report_margin
end if

start_print_time = time(0)
key_input$ = ''
line_counter%   = lines_per_page%  ! cause first page break
first_line%   = true

end routine


!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! P R I N T   R E P O R T:
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
routine print_report
```

```
        LET HOURS = 0
        TOTAL(1,LEVEL%) = 0
        TOTAL(2,LEVEL%) = 0
        TOTAL(3,LEVEL%) = 0
13020   for each MRR
          gosub show_report_status
          if  bailout% = true  then exit for
          ! Do file lookups

        print toolno, hours, mrr ( fab_cost )

        HOURS = HOURS + MRR ( FAB_COST )

          gosub set_user_defined
          gosub add_totals
          gosub print_detail_line
          first_line% = false
          level_count(0) = level_count(0) + 1
        next MRR
        level% = 0
        gosub print_totals
        if  out_ch > 0  then close #out_ch

13099   end routine


13500   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! R E L A T E   M A N Y
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine relate_many


        end routine


14000   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! D I S P L A Y   R E P O R T
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        !       u_dispatch$ = routine to dispatch to after clear screen
        !                     (u_dispatch$ is cleared on reference)
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine display_report

14020   if  not making_report  then
          gosub show_stats
          delay
          exit routine
        end if
        if  out_ch = 0  then exit routine
        set system : comment "Displaying report"
        u_str$ = report_name$
        u_scr_width% = margin
        u_dispatch$ = 'show_stats'
        gosub prnt_ask_option

14099   end routine


21000   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! N E W   P A G E
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine new_page
```

```
21020   if  not making_report  then  exit routine
        page_counter% = page_counter% + 1
        gosub show_recs_printed
        gosub show_pages_printed
        gosub show_eoj
        print #out_ch: chr$(12);

        gosub check_title_substitutes

        page$ = "Page " + str$(page_counter%)
        p% = len(page$)
        t% = len(title$(1))
        rpt_width% = max(report_width%,t%+22%)
        rpt_width% = max(rpt_width%, len(title$(2)))
        rpt_width% = max(rpt_width%, len(title$(3)))

        z% = (rpt_width% - t%) / 2
        if  z% < 13%  then  z% = 13% ! leave room for the date
        print #out_ch: today$;
          print #out_ch: tab(z% + 1 ); title$(1);
          print #out_ch: tab(rpt_width% - p% + 1); page$; chr$(0)
          line_counter% = 1

        print #out_ch:tab(1); "SPECIAL TOOLING CARD";
        print #out_ch:tab(114); "COST HISTORY CARD"
        for i% = 2 to titles%
          z% = (rpt_width% - len(title$(i%)))/2
          if  z% < 1%  then  z% = 1%
          print #out_ch: tab(z% + 1); title$(i%); chr$(0)
          line_counter% = line_counter% + 1
        next i%

        print #out_ch:tab(2); "TOOL NBR:";  TAB(13); MRR(TOOLNO)

        print #out_ch: chr$(0)
        line_counter% = line_counter% + 1

        print #out_ch : tab(1); "DATE";
        print #out_ch : tab(10); "TWO ";
        print #out_ch : tab(21); "CCN";
        print #out_ch : tab(32); "D E S C R I P T I O N";
        print #out_ch : tab(88); "MAT_COST";
        print #out_ch : tab(103); "FAB_COST";
        print #out_ch : tab(117); "TOTAL";
        print #out_ch :  chr$(0)
        line_counter% = line_counter% + 1
        print #out_ch : tab(1);  repeat$("-", 8);
        print #out_ch : tab(10); repeat$("-", 10);
        print #out_ch : tab(21); repeat$("-", 10);
        print #out_ch : tab(32); repeat$("-", 50);
        print #out_ch : tab(88); repeat$("-", 12);
        print #out_ch : tab(103); repeat$("-", 12);
        print #out_ch : tab(117); repeat$("-", 12);
        print #out_ch :  chr$(0)
        line_counter% = line_counter% + 1
        first_page_line% = true
21099   end routine


21100   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! C H E C K   T I T L E   S U B S T I T U T E S
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine check_title_substitutes
```

```
21120   when exception in
        use
          continue
        end when


21199   end routine


22000   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! S E T   U S E R   D E F I N E D
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine set_user_defined

22020   ! User-defined variables get calculated here
        when exception in
        use
          if  extype = no_current_record  then continue
        end when
22099   end routine

24000   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! A D D   T O T A L S
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine add_totals

24020   when exception in
          for level% = 0 to levels%
            ! Add level totals here
            total(1,level%) = total(1,level%) + MRR(MAT_COST)
            total(2,level%) = total(2,level%) + MRR(FAB_COST)
            total(3,level%) = total(3,level%) + MRR(TOTAL)
          next level%
        use
          if  extype = no_current_record  then continue
        end when
24099   end routine


26000   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! P R I N T   D E T A I L   L I N E
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! Print detail line: col_data$(col) for each column
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine print_detail_line

26020   ! Print detail lines here
        if  line_counter% >= lines_per_page%  then gosub new_page
        data_printed% = false
        when exception in
        col_len% = 8
          print #out_ch : tab(1);
          print #out_ch, using "%%~-%%~-%%" : MRR ( DATE );
          data_printed% = true

        col_len% = 10
          print #out_ch : tab(10);
          print #out_ch, using "<#########" : MRR ( TWO );
          data_printed% = true

        col_len% = 10
          print #out_ch : tab(21);
          print #out_ch, using "<#########" : MRR ( CCN );
          data_printed% = true
```

```
        col_len% = 50
          print #out_ch : tab(32);
          print #out_ch, using "<################################################
##" : MRR ( CHG_DESC );
          data_printed% = true

        col_len% = 12
          print #out_ch : tab(88);
          print #out_ch, using "$##,###,###" : MRR ( MAT_COST );
          data_printed% = true

        col_len% = 12
          print #out_ch : tab(102);
          print #out_ch : "HRS" ; MRR ( FAB_COST );
          data_printed% = true

        col_len% = 12
          print #out_ch : tab(117);
          print #out_ch, using "$##,###,###" : MRR ( TOTAL );
          data_printed% = true

          print #out_ch :
          line_counter% = line_counter% + 1

        use
          print #out_ch: repeat$(lookup_err$, col_len%);
          continue
        end when

        for z% = 1 to spacing%
          print #out_ch:
          line_counter% = line_counter% + 1
        next z%

        data_printed% = false
        first_page_line% = false
        last_break_level% = 0

26099   end routine

27000   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! P R I N T   T O T A L S
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        routine print_totals

27020   if  tsuppress% and (level_count(level%) = 1) and (level% = levels%) &
            then exit routine
        if  skip_break_totals(level%) then exit routine
        if  line_counter% >= lines_per_page%  then gosub new_page

        print #out_ch : tab(88); repeat$("=", 12);
        print #out_ch : tab(103); repeat$("=", 12);
        print #out_ch : tab(117); repeat$("=", 12);
        print #out_ch :
        line_counter% = line_counter% + 1

        if  break_pos(level%) <> 0 then &
          print #out_ch : tab(break_pos(level%)); break_desc$(level%);
        when exception in
          col_len% = 12
          print #out_ch : tab(87);
          print #out_ch, using "$###,###,###" : TOTAL(1,LEVEL%);

          col_len% = 12
          print #out_ch : tab(102);
          print #out_ch : "HRS" ; tab(106) ; HOURS;
```

```
          col_len% = 12
          print #out_ch : tab(116);
          print #out_ch, using "$###,###,###" : TOTAL(3,LEVEL%);

        use
          print #out_ch: repeat$(lookup_err$, col_len%);
          continue
        end when

        print #out_ch:
        line_counter% = line_counter% + 1

27099   end routine

81600   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! A S K
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        !
        ! Ask the expected prompt.
        !
        ! EXPECTED:
        !       u_prompt$ = question being asked
        !       u_default$ = the default
        !       u_len = the length.  Doesn't reset any of these.
        !       u_help_key$ = the help key (optional; will use the prompt
        !          with spaces changed to _ and "?:" removed)
        !       u_required = true if a response is necessary
        !
        ! RESULT:
        !       u_reply$  = user's reply
        !       u_help+key$ is reset to ""
        !       u_required = true
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  ask:

81620   clear area 21,1,21,old_margin
        if  u_help_key$ = ""  then &
          u_help_key$ = change$(trim$(change$(u_prompt$,'?:'))," ","_")
        do

        if u_default$ <> "" then toolno$ = u_default$
        if u_default$ = "" then u_default$ = toolno$
        set system, symbol 'otms_toolno$':value toolno$
!*added three lines above        !reset toolno$ to u_default$ and
                                 !set global symbol equal to toolno$
                                 !reset u_default$ to toolno$

          u_reply$ = ""
          line input at 21,1, &
                     prompt u_prompt$, &
                     default u_default$, &
                     length u_len: u_reply$
          if  _help  then
            h_key$ = u_help_key$
!            gosub h_help
            if  u_len = 0  then clear area 21,1,21,old_margin
            repeat do
          end if
          if  _exit or _back  then exit do
          if  u_reply$ = "" and u_required  then  repeat do
          clear area 21,1,21,old_margin

        end do
        u_help_key$ = ""
```

```
        u_required = true

81699   return

83000   go to 11200

90000   %include "usr_root:[software.intouch]guide_shell_intl.inc"

99999   end
```

Exhibit D

```
Directory D4:[PRODUCTION.OTMS.SRC]

B.COM;2                        1    8-JUN-1994 13:26:20.35
BARCODE.DIR;1                  1    6-APR-1996 09:30:26.76
CENT.INT;135                   6    4-MAY-1996 13:32:37.86
CONTRACT_NO.INT;8              5    6-APR-1996 09:52:46.17
CONVERT1.INT;2                 2    5-OCT-1996 12:39:48.66
CONVERT2.INT;2                 2   25-OCT-1996 08:23:26.72
CONVERT4.INT;1                 5   17-OCT-1996 12:03:09.65
CONVERT5.INT;3                 5   25-OCT-1996 14:40:23.63
FIXMRR.INT;2                  42    6-FEB-1995 14:30:03.82 *
GOV.DEF;1                     27    8-JUN-1994 07:37:56.16
GOV.STR;1                      1    8-JUN-1994 07:28:37.26
GOVMOD.INT;7                  39   15-JUL-1995 13:02:36.61
GOVMODSUB.INT;7               44   30-MAY-1995 12:27:59.16
GOVRPT.INT;8                  38   15-JUL-1995 13:03:14.65
GOVSPMOD.INT;8                55   30-MAY-1995 12:28:51.47
GOVTPSUB.INT;8                43   30-MAY-1995 12:29:43.84
INAMOD.INT;2                  38   20-OCT-1995 10:19:43.40
INASPMOD.INT;2                56    3-NOV-1995 13:47:33.33
INATOL.INT;2                  39   20-OCT-1995 10:20:38.18
INQMRR_MAIN.INT;2            316   18-SEP-1995 12:30:39.88
INQOTMS000.INT;4              12   19-SEP-1995 07:55:37.40
INQPART_MAIN.INT;2           315   18-SEP-1995 12:34:55.08
INQRR_MAIN.INT;3             316   18-SEP-1995 12:38:22.09
INQTOOL_MAIN.INT;1           318   18-SEP-1995 12:04:50.50
INQTXT_MAIN.INT;2            316   18-SEP-1995 12:42:11.01
LAST_INV.DAT;6               539    4-MAY-1996 13:21:42.93
MAINTAIN.INT;1               315   10-SEP-1992 11:16:26.20
MODELRPT.INT;10               35    3-JUN-1993 09:25:56.49
MODSUB1.INT;40                40    3-JUN-1993 12:48:05.47
MOD_MASS.DAT;5                 0    6-APR-1996 09:52:52.46
MRR.DEF;1                     27   16-JUL-1994 13:04:00.40
MRR.INT;80                    42   15-JUL-1995 12:57:52.69
MRR.MAIN;33                   14    2-SEP-1994 12:30:38.82
MRR.STR;3                      1   20-SEP-1994 13:21:53.77
MRR_MAIN.INT;7               316   20-JUL-1995 14:37:44.68
O.LIS;4                        7   25-OCT-1996 15:25:58.51
OTMS000.INT;31                13   25-OCT-1996 10:57:15.59
OTMS_SRC_DIR.STR;1             1   21-OCT-1995 11:36:39.85
OUT.LIS;8                   1381    6-APR-1996 09:21:24.28
PART2.DEF;1                   27   16-OCT-1996 10:09:41.12
PART2.FDL;2                    4   17-OCT-1996 07:52:33.67
PART2.MAIN;8                  13   25-OCT-1996 10:21:33.98
PART2.STR;1                    1   16-OCT-1996 09:32:41.15
PARTMOD.INT;3                 35   15-JUL-1995 13:01:23.06
PARTMOD2.INT;18               36    9-NOV-1996 09:34:08.32
PARTMODSUB.INT;4              40    9-NOV-1996 10:08:49.12
PARTMODSUB2.INT;9             40    9-NOV-1996 10:11:21.45
PARTRPT.INT;11                35    9-NOV-1996 10:02:18.37
PARTRPT.INT_IMG;1            854    9-NOV-1996 10:03:09.35
PARTRPT2.INT;8                35    9-NOV-1996 10:05:29.92
PARTS.DAT;1                  144    7-JUN-1994 15:49:13.99
PARTS.DEF;2                   27    7-JUN-1994 15:41:34.28
PARTS.FDL;2                    4    5-OCT-1996 11:44:30.77
PARTS.MAIN;2                  13    5-OCT-1996 12:32:23.86
PARTS.STR;1                    1    7-JUN-1994 15:42:48.74
PARTSPMOD.INT;5               51   12-OCT-1996 10:45:04.97
PARTSPMOD.INT_IMG;1
                             854   25-OCT-1996 10:52:31.13
PARTSPMOD2.INT;49             52    9-NOV-1996 10:53:21.70
PARTSUB.INT;3                 40   30-MAY-1995 12:25:44.86
PARTSUB2.INT;12               40    9-NOV-1996 11:00:17.56
PART_MAIN.INT;4              315   17-OCT-1996 13:13:23.64
REPORT1.GUIDE;1                1   25-SEP-1992 12:20:14.38
```

Exhibit E

ALL-STATE® LEGAL 800-222-0510 ED811 RECYCLED

```
1       !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! Program:      Shop Day Generator Utility
        ! Package:      Macpac/D Control File
        ! Author :      Bruce Dickens
        ! Date   :      March 13, 1995
        !Purpose :      Provide a method to generate 9998
        !               contol file records for any year.
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


1000    !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! M A I N   L O G I C   S E C T I O N
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        !
        ! Ask for any calendar year and generate all MacPac/D
        ! 9998 control file records for that year for input into
        ! the MacPac\D update facility.
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        input 'Calendar Year to Generate? (YYYY)': Year%        !BDickens000001
        open #1:name 'calgen.lis', access output                !BDickens000002
                    option base 0                               !BDickens000003
                    dim wkdyr$(28)                              !BDickens000004
                    wkdyr$(0)  = "fri"                          !BDickens000005
                    wkdyr$(1)  = "sun"                          !BDickens000006
                    wkdyr$(2)  = "mon"                          !BDickens000007
                    wkdyr$(3)  = "tue"                          !BDickens000008
                    wkdyr$(4)  = "wed"                          !BDickens000009
                    wkdyr$(5)  = "fri"                          !BDickens000010
                    wkdyr$(6)  = "sat"                          !BDickens000011
                    wkdyr$(7)  = "sun"                          !BDickens000012
                    wkdyr$(8)  = "mon"                          !BDickens000013
                    wkdyr$(9)  = "wed"                          !BDickens000014
                    wkdyr$(10)  = "thu"                         !BDickens000015
                    wkdyr$(11)  = "fri"                         !BDickens000016
                    wkdyr$(12)  = "sat"                         !BDickens000017
                    wkdyr$(13)  = "mon"                         !BDickens000018
                    wkdyr$(14)  = "tue"                         !BDickens000019
                    wkdyr$(15)  = "wed"                         !BDickens000020
                    wkdyr$(16)  = "thu"                         !BDickens000021
                    wkdyr$(17)  = "sat"                         !BDickens000022
                    wkdyr$(18)  = "sun"                         !BDickens000023
                    wkdyr$(19)  = "mon"                         !BDickens000024
                    wkdyr$(20)  = "tue"                         !BDickens000025
                    wkdyr$(21)  = "thu"                         !BDickens000026
                    wkdyr$(22)  = "fri"                         !BDickens000027
                    wkdyr$(23)  = "sat"                         !BDickens000028
                    wkdyr$(24)  = "sun"                         !BDickens000029
                    wkdyr$(25)  = "tue"                         !BDickens000030
                    wkdyr$(26)  = "wed"                         !BDickens000031
                    wkdyr$(27)  = "thu"                         !BDickens000032
                    wkdyr$(28)  = "fri"                         !BDickens000033
        a = year%/28                                            !BDickens000034
        c = mod(year%,28)                                       !BDickens000035
          b = year%/4                                           !BDickens000036
          d = mod (year%,4)                                     !BDickens000037
          if d = 0 then                                         !BDickens000038
            e = 12                                              !BDickens000039
          else                                                  !BDickens000040
            e = 11                                              !BDickens000041
          end if                                                !BDickens000042

        days = 0                                                !BDickens000043
        f = c                                                   !BDickens000044

 5000           option base 0                                   !BDickens000045
```

```
        dim wkd$(7)                                    !BDickens000046
            wkd$(0)  =  "fri"                          !BDickens000047
            wkd$(1)  =  "sat"                          !BDickens000048
            wkd$(2)  =  "sun"                          !BDickens000049
            wkd$(3)  =  "mon"                          !BDickens000050
            wkd$(4)  =  "tue"                          !BDickens000051
            wkd$(5)  =  "wed"                          !BDickens000052
            wkd$(6)  =  "thu"                          !BDickens000053
    for g = 0 to 6                                     !BDickens000054
      if wkd$(g) = wkdyr$(f) then i = g                !BDickens000055
    next g                                             !BDickens000056

                                                       !BDickens000057
6000    dim mo$(12)                                    !BDickens000058
            mo$(1)  =   "JAN"                          !BDickens000058
            mo$(2)  =   "FEB"                          !BDickens000059
            mo$(3)  =   "MAR"                          !BDickens000060
            mo$(4)  =   "APR"                          !BDickens000061
            mo$(5)  =   "MAY"                          !BDickens000062
            mo$(6)  =   "JUN"                          !BDickens000063
            mo$(7)  =   "JUL"                          !BDickens000064
            mo$(8)  =   "AUG"                          !BDickens000065
            mo$(9)  =   "SEP"                          !BDickens000066
            mo$(10)  =  "OCT"                          !BDickens000067
            mo$(11)  =  "NOV"                          !BDickens000068
            mo$(12)  =  "DEC"


                                                       !BDickens000069
    day% = 0                                           !BDickens000070
    mo% = 0                                            !BDickens000071
    shopday% = 0                                       !BDickens000072
    count% = 0

                                                       !BDickens000073
    do until mo% = 12                                  !BDickens000074
      mo% = mo% + 1                                    !BDickens000075
        for day% = 1 to 31                             !BDickens000076
          h = mod(i,7)                                 !BDickens000077
          cal$ = str$(year%)  + lpad$(str$(mo%),2,'0') &  !BDickens000078
                              + lpad$(str$(day%),2,'0')


12000   !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! Holiday Section
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ! This section of code assigns shop days and lists
        ! calendar days.
        !%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

          if d = 0 then
             goto 12100
          else
            if cal$ = str$(year%) + "0229"  then goto 13500
          end if

12100       if cal$ = str$(year%)  + "0230"  then goto 13500
            if cal$ = str$(year%)  + "0231"  then goto 13500
            if cal$ = str$(year%)  + "0431"  then goto 13500
            if cal$ = str$(year%)  + "0631"  then goto 13500
            if cal$ = str$(year%)  + "0931"  then goto 13500
            if cal$ = str$(year%)  + "1131"  then goto 13500


            if wkd$(h) = "sun" and cal$ = str$(year%) + "0101" then
                shopday% = 1
                goto 13000
```

```
        if wkd$(h) = "mon" and cal$ = str$(year%) + "0907" then goto 13000

!           Thanksgiving (4th Thursday in November)
        if wkd$(h) = "thu" and cal$ = str$(year%) + "1122" then goto 13000
        if wkd$(h) = "fri" and cal$ = str$(year%) + "1123" then goto 13000
        if wkd$(h) = "thu" and cal$ = str$(year%) + "1123" then goto 13000
        if wkd$(h) = "fri" and cal$ = str$(year%) + "1124" then goto 13000
        if wkd$(h) = "thu" and cal$ = str$(year%) + "1124" then goto 13000
        if wkd$(h) = "fri" and cal$ = str$(year%) + "1125" then goto 13000
        if wkd$(h) = "thu" and cal$ = str$(year%) + "1125" then goto 13000
        if wkd$(h) = "fri" and cal$ = str$(year%) + "1126" then goto 13000
        if wkd$(h) = "thu" and cal$ = str$(year%) + "1126" then goto 13000
        if wkd$(h) = "fri" and cal$ = str$(year%) + "1127" then goto 13000
        if wkd$(h) = "thu" and cal$ = str$(year%) + "1127" then goto 13000
        if wkd$(h) = "fri" and cal$ = str$(year%) + "1128" then goto 13000
        if wkd$(h) = "thu" and cal$ = str$(year%) + "1128" then goto 13000
        if wkd$(h) = "fri" and cal$ = str$(year%) + "1129" then goto 13000

!           Christmas Break (December 25th)
        if wkd$(h) = "fri" and cal$ = str$(year%) + "1222" then goto 13000
        if wkd$(h) = "mon" and cal$ = str$(year%) + "1223" then goto 13000
        if cal$ = str$(year%) + "1224"  then goto 13000
        if cal$ = str$(year%) + "1225"  then goto 13000
        if cal$ = str$(year%) + "1226"  then goto 13000
        if cal$ = str$(year%) + "1227"  then goto 13000
        if cal$ = str$(year%) + "1228"  then goto 13000
        if cal$ = str$(year%) + "1229"  then goto 13000
        if cal$ = str$(year%) + "1230"  then goto 13000
        if cal$ = str$(year%) + "1231"  then goto 13000

12500     shopday% = shopday% + 1

13000   i = i + 1
          count% = count% + 1
            k$ = cal$[3:4] + lpad$(str$(shopday%),3,'0')
            m$ = mo$(mo%) + ' ' + cal$[7:8] + ', ' + cal$[1:4]
            print #1:  '99982'; cal$;'      '; cal$[3:8]; k$ ;  &
                      '                 ';m$; '                 '; 'C'

13500     next day%                                          !BDickens000079
        loop                                                 !BDickens000080

99999   end                                                  !BDickens000081
```

9998219960512	96051296094	MAY 12, 1996	C
9998219960513	96051396095	MAY 13, 1996	C
9998219960514	96051496096	MAY 14, 1996	C
9998219960515	96051596097	MAY 15, 1996	C
9998219960516	96051696098	MAY 16, 1996	C
9998219960517	96051796099	MAY 17, 1996	C
9998219960518	96051896099	MAY 18, 1996	C
9998219960519	96051996099	MAY 19, 1996	C
9998219960520	96052096100	MAY 20, 1996	C
9998219960521	96052196101	MAY 21, 1996	C
9998219960522	96052296102	MAY 22, 1996	C
9998219960523	96052396103	MAY 23, 1996	C
9998219960524	96052496104	MAY 24, 1996	C
9998219960525	96052596104	MAY 25, 1996	C
9998219960526	96052696104	MAY 26, 1996	C
9998219960527	96052796104	MAY 27, 1996	C
9998219960528	96052896105	MAY 28, 1996	C
9998219960529	96052996106	MAY 29, 1996	C
9998219960530	96053096107	MAY 30, 1996	C
9998219960531	96053196108	MAY 31, 1996	C
9998219960601	96060196108	JUN 01, 1996	C
9998219960602	96060296108	JUN 02, 1996	C
9998219960603	96060396109	JUN 03, 1996	C
9998219960604	96060496110	JUN 04, 1996	C
9998219960605	96060596111	JUN 05, 1996	C
9998219960606	96060696112	JUN 06, 1996	C
9998219960607	96060796113	JUN 07, 1996	C
9998219960608	96060896113	JUN 08, 1996	C
9998219960609	96060996113	JUN 09, 1996	C
9998219960610	96061096114	JUN 10, 1996	C
9998219960611	96061196115	JUN 11, 1996	C
9998219960612	96061296116	JUN 12, 1996	C
9998219960613	96061396117	JUN 13, 1996	C
9998219960614	96061496118	JUN 14, 1996	C
9998219960615	96061596118	JUN 15, 1996	C
9998219960616	96061696118	JUN 16, 1996	C
9998219960617	96061796119	JUN 17, 1996	C
9998219960618	96061896120	JUN 18, 1996	C
9998219960619	96061996121	JUN 19, 1996	C
9998219960620	96062096122	JUN 20, 1996	C
9998219960621	96062196123	JUN 21, 1996	C
9998219960622	96062296123	JUN 22, 1996	C
9998219960623	96062396123	JUN 23, 1996	C
9998219960624	96062496124	JUN 24, 1996	C
9998219960625	96062596125	JUN 25, 1996	C
9998219960626	96062696126	JUN 26, 1996	C
9998219960627	96062796127	JUN 27, 1996	C
9998219960628	96062896128	JUN 28, 1996	C
9998219960629	96062996128	JUN 29, 1996	C
9998219960630	96063096128	JUN 30, 1996	C
9998219960701	96070196129	JUL 01, 1996	C
9998219960702	96070296130	JUL 02, 1996	C
9998219960703	96070396131	JUL 03, 1996	C
9998219960704	96070496131	JUL 04, 1996	C
9998219960705	96070596132	JUL 05, 1996	C
9998219960706	96070696132	JUL 06, 1996	C
9998219960707	96070796132	JUL 07, 1996	C
9998219960708	96070896133	JUL 08, 1996	C
9998219960709	96070996134	JUL 09, 1996	C
9998219960710	96071096135	JUL 10, 1996	C
9998219960711	96071196136	JUL 11, 1996	C
9998219960712	96071296137	JUL 12, 1996	C
9998219960713	96071396137	JUL 13, 1996	C
9998219960714	96071496137	JUL 14, 1996	C
9998219960715	96071596138	JUL 15, 1996	C
9998219960716	96071696139	JUL 16, 1996	C

9998219960921	96092196186	SEP 21, 1996	C
9998219960922	96092296186	SEP 22, 1996	C
9998219960923	96092396187	SEP 23, 1996	C
9998219960924	96092496188	SEP 24, 1996	C
9998219960925	96092596189	SEP 25, 1996	C
9998219960926	96092696190	SEP 26, 1996	C
9998219960927	96092796191	SEP 27, 1996	C
9998219960928	96092896191	SEP 28, 1996	C
9998219960929	96092996191	SEP 29, 1996	C
9998219960930	96093096192	SEP 30, 1996	C
9998219961001	96100196193	OCT 01, 1996	C
9998219961002	96100296194	OCT 02, 1996	C
9998219961003	96100396195	OCT 03, 1996	C
9998219961004	96100496196	OCT 04, 1996	C
9998219961005	96100596196	OCT 05, 1996	C
9998219961006	96100696196	OCT 06, 1996	C
9998219961007	96100796197	OCT 07, 1996	C
9998219961008	96100896198	OCT 08, 1996	C
9998219961009	96100996199	OCT 09, 1996	C
9998219961010	96101096200	OCT 10, 1996	C
9998219961011	96101196201	OCT 11, 1996	C
9998219961012	96101296201	OCT 12, 1996	C
9998219961013	96101396201	OCT 13, 1996	C
9998219961014	96101496202	OCT 14, 1996	C
9998219961015	96101596203	OCT 15, 1996	C
9998219961016	96101696204	OCT 16, 1996	C
9998219961017	96101796205	OCT 17, 1996	C
9998219961018	96101896206	OCT 18, 1996	C
9998219961019	96101996206	OCT 19, 1996	C
9998219961020	96102096206	OCT 20, 1996	C
9998219961021	96102196207	OCT 21, 1996	C
9998219961022	96102296208	OCT 22, 1996	C
9998219961023	96102396209	OCT 23, 1996	C
9998219961024	96102496210	OCT 24, 1996	C
9998219961025	96102596211	OCT 25, 1996	C
9998219961026	96102696211	OCT 26, 1996	C
9998219961027	96102796211	OCT 27, 1996	C
9998219961028	96102896212	OCT 28, 1996	C
9998219961029	96102996213	OCT 29, 1996	C
9998219961030	96103096214	OCT 30, 1996	C
9998219961031	96103196215	OCT 31, 1996	C
9998219961101	96110196216	NOV 01, 1996	C
9998219961102	96110296216	NOV 02, 1996	C
9998219961103	96110396216	NOV 03, 1996	C
9998219961104	96110496217	NOV 04, 1996	C
9998219961105	96110596218	NOV 05, 1996	C
9998219961106	96110696219	NOV 06, 1996	C
9998219961107	96110796220	NOV 07, 1996	C
9998219961108	96110896221	NOV 08, 1996	C
9998219961109	96110996221	NOV 09, 1996	C
9998219961110	96111096221	NOV 10, 1996	C
9998219961111	96111196222	NOV 11, 1996	C
9998219961112	96111296223	NOV 12, 1996	C
9998219961113	96111396224	NOV 13, 1996	C
9998219961114	96111496225	NOV 14, 1996	C
9998219961115	96111596226	NOV 15, 1996	C
9998219961116	96111696226	NOV 16, 1996	C
9998219961117	96111796226	NOV 17, 1996	C
9998219961118	96111896227	NOV 18, 1996	C
9998219961119	96111996228	NOV 19, 1996	C
9998219961120	96112096229	NOV 20, 1996	C
9998219961121	96112196230	NOV 21, 1996	C
9998219961122	96112296231	NOV 22, 1996	C
9998219961123	96112396231	NOV 23, 1996	C
9998219961124	96112496231	NOV 24, 1996	C
9998219961125	96112596232	NOV 25, 1996	C

9998219961126	96112696233	NOV 26, 1996	C
9998219961127	96112796234	NOV 27, 1996	C
9998219961128	96112896234	NOV 28, 1996	C
9998219961129	96112996234	NOV 29, 1996	C
9998219961130	96113096234	NOV 30, 1996	C
9998219961201	96120196234	DEC 01, 1996	C
9998219961202	96120296235	DEC 02, 1996	C
9998219961203	96120396236	DEC 03, 1996	C
9998219961204	96120496237	DEC 04, 1996	C
9998219961205	96120596238	DEC 05, 1996	C
9998219961206	96120696239	DEC 06, 1996	C
9998219961207	96120796239	DEC 07, 1996	C
9998219961208	96120896239	DEC 08, 1996	C
9998219961209	96120996240	DEC 09, 1996	C
9998219961210	96121096241	DEC 10, 1996	C
9998219961211	96121196242	DEC 11, 1996	C
9998219961212	96121296243	DEC 12, 1996	C
9998219961213	96121396244	DEC 13, 1996	C
9998219961214	96121496244	DEC 14, 1996	C
9998219961215	96121596244	DEC 15, 1996	C
9998219961216	96121696245	DEC 16, 1996	C
9998219961217	96121796246	DEC 17, 1996	C
9998219961218	96121896247	DEC 18, 1996	C
9998219961219	96121996248	DEC 19, 1996	C
9998219961220	96122096249	DEC 20, 1996	C
9998219961221	96122196249	DEC 21, 1996	C
9998219961222	96122296249	DEC 22, 1996	C
9998219961223	96122396249	DEC 23, 1996	C
9998219961224	96122496249	DEC 24, 1996	C
9998219961225	96122596249	DEC 25, 1996	C
9998219961226	96122696249	DEC 26, 1996	C
9998219961227	96122796249	DEC 27, 1996	C
9998219961228	96122896249	DEC 28, 1996	C
9998219961229	96122996249	DEC 29, 1996	C
9998219961230	96123096249	DEC 30, 1996	C
9998219961231	96123196249	DEC 31, 1996	C

9998219960307	96030796048	MAR 07, 1996	C
9998219960308	96030896049	MAR 08, 1996	C
9998219960309	96030996049	MAR 09, 1996	C
9998219960310	96031096049	MAR 10, 1996	C
9998219960311	96031196050	MAR 11, 1996	C
9998219960312	96031296051	MAR 12, 1996	C
9998219960313	96031396052	MAR 13, 1996	C
9998219960314	96031496053	MAR 14, 1996	C
9998219960315	96031596054	MAR 15, 1996	C
9998219960316	96031696054	MAR 16, 1996	C
9998219960317	96031796054	MAR 17, 1996	C
9998219960318	96031896055	MAR 18, 1996	C
9998219960319	96031996056	MAR 19, 1996	C
9998219960320	96032096057	MAR 20, 1996	C
9998219960321	96032196058	MAR 21, 1996	C
9998219960322	96032296059	MAR 22, 1996	C
9998219960323	96032396059	MAR 23, 1996	C
9998219960324	96032496059	MAR 24, 1996	C
9998219960325	96032596060	MAR 25, 1996	C
9998219960326	96032696061	MAR 26, 1996	C
9998219960327	96032796062	MAR 27, 1996	C
9998219960328	96032896063	MAR 28, 1996	C
9998219960329	96032996064	MAR 29, 1996	C
9998219960330	96033096064	MAR 30, 1996	C
9998219960331	96033196064	MAR 31, 1996	C
9998219960401	96040196065	APR 01, 1996	C
9998219960402	96040296066	APR 02, 1996	C
9998219960403	96040396067	APR 03, 1996	C
9998219960404	96040496068	APR 04, 1996	C
9998219960405	96040596069	APR 05, 1996	C
9998219960406	96040696069	APR 06, 1996	C
9998219960407	96040796069	APR 07, 1996	C
9998219960408	96040896070	APR 08, 1996	C
9998219960409	96040996071	APR 09, 1996	C
9998219960410	96041096072	APR 10, 1996	C
9998219960411	96041196073	APR 11, 1996	C
9998219960412	96041296074	APR 12, 1996	C
9998219960413	96041396074	APR 13, 1996	C
9998219960414	96041496074	APR 14, 1996	C
9998219960415	96041596075	APR 15, 1996	C
9998219960416	96041696076	APR 16, 1996	C
9998219960417	96041796077	APR 17, 1996	C
9998219960418	96041896078	APR 18, 1996	C
9998219960419	96041996079	APR 19, 1996	C
9998219960420	96042096079	APR 20, 1996	C
9998219960421	96042196079	APR 21, 1996	C
9998219960422	96042296080	APR 22, 1996	C
9998219960423	96042396081	APR 23, 1996	C
9998219960424	96042496082	APR 24, 1996	C
9998219960425	96042596083	APR 25, 1996	C
9998219960426	96042696084	APR 26, 1996	C
9998219960427	96042796084	APR 27, 1996	C
9998219960428	96042896084	APR 28, 1996	C
9998219960429	96042996085	APR 29, 1996	C
9998219960430	96043096086	APR 30, 1996	C
9998219960501	96050196087	MAY 01, 1996	C
9998219960502	96050296088	MAY 02, 1996	C
9998219960503	96050396089	MAY 03, 1996	C
9998219960504	96050496089	MAY 04, 1996	C
9998219960505	96050596089	MAY 05, 1996	C
9998219960506	96050696090	MAY 06, 1996	C
9998219960507	96050796091	MAY 07, 1996	C
9998219960508	96050896092	MAY 08, 1996	C
9998219960509	96050996093	MAY 09, 1996	C
9998219960510	96051096094	MAY 10, 1996	C
9998219960511	96051196094	MAY 11, 1996	C

9998219960101	96010196001	JAN 01, 1996	C
9998219960102	96010296001	JAN 02, 1996	C
9998219960103	96010396002	JAN 03, 1996	C
9998219960104	96010496003	JAN 04, 1996	C
9998219960105	96010596004	JAN 05, 1996	C
9998219960106	96010696004	JAN 06, 1996	C
9998219960107	96010796004	JAN 07, 1996	C
9998219960108	96010896005	JAN 08, 1996	C
9998219960109	96010996006	JAN 09, 1996	C
9998219960110	96011096007	JAN 10, 1996	C
9998219960111	96011196008	JAN 11, 1996	C
9998219960112	96011296009	JAN 12, 1996	C
9998219960113	96011396009	JAN 13, 1996	C
9998219960114	96011496009	JAN 14, 1996	C
9998219960115	96011596010	JAN 15, 1996	C
9998219960116	96011696011	JAN 16, 1996	C
9998219960117	96011796012	JAN 17, 1996	C
9998219960118	96011896013	JAN 18, 1996	C
9998219960119	96011996014	JAN 19, 1996	C
9998219960120	96012096014	JAN 20, 1996	C
9998219960121	96012196014	JAN 21, 1996	C
9998219960122	96012296015	JAN 22, 1996	C
9998219960123	96012396016	JAN 23, 1996	C
9998219960124	96012496017	JAN 24, 1996	C
9998219960125	96012596018	JAN 25, 1996	C
9998219960126	96012696019	JAN 26, 1996	C
9998219960127	96012796019	JAN 27, 1996	C
9998219960128	96012896019	JAN 28, 1996	C
9998219960129	96012996020	JAN 29, 1996	C
9998219960130	96013096021	JAN 30, 1996	C
9998219960131	96013196022	JAN 31, 1996	C
9998219960201	96020196023	FEB 01, 1996	C
9998219960202	96020296024	FEB 02, 1996	C
9998219960203	96020396024	FEB 03, 1996	C
9998219960204	96020496024	FEB 04, 1996	C
9998219960205	96020596025	FEB 05, 1996	C
9998219960206	96020696026	FEB 06, 1996	C
9998219960207	96020796027	FEB 07, 1996	C
9998219960208	96020896028	FEB 08, 1996	C
9998219960209	96020996029	FEB 09, 1996	C
9998219960210	96021096029	FEB 10, 1996	C
9998219960211	96021196029	FEB 11, 1996	C
9998219960212	96021296030	FEB 12, 1996	C
9998219960213	96021396031	FEB 13, 1996	C
9998219960214	96021496032	FEB 14, 1996	C
9998219960215	96021596033	FEB 15, 1996	C
9998219960216	96021696034	FEB 16, 1996	C
9998219960217	96021796034	FEB 17, 1996	C
9998219960218	96021896034	FEB 18, 1996	C
9998219960219	96021996035	FEB 19, 1996	C
9998219960220	96022096036	FEB 20, 1996	C
9998219960221	96022196037	FEB 21, 1996	C
9998219960222	96022296038	FEB 22, 1996	C
9998219960223	96022396039	FEB 23, 1996	C
9998219960224	96022496039	FEB 24, 1996	C
9998219960225	96022596039	FEB 25, 1996	C
9998219960226	96022696040	FEB 26, 1996	C
9998219960227	96022796041	FEB 27, 1996	C
9998219960228	96022896042	FEB 28, 1996	C
9998219960229	96022996043	FEB 29, 1996	C
9998219960301	96030196044	MAR 01, 1996	C
9998219960302	96030296044	MAR 02, 1996	C
9998219960303	96030396044	MAR 03, 1996	C
9998219960304	96030496045	MAR 04, 1996	C
9998219960305	96030596046	MAR 05, 1996	C
9998219960306	96030696047	MAR 06, 1996	C

Attachment 1

Menu request:

```
BRUCED> sint calgen
Calendar Year to Generate? (YYYY)? 1996
```

1. CALGEN is the executable.
2. Calendar Year to Generate? (YYYY)?   is the prompt.
3. 1996 is the reply.

The following pages are the formatted output for input to the MacPac control file and calendar update programs. I would prefer to have my algorithm put in place of all those pages of 9998 records.

The program is designed very simply. The intouch code is less than four pages long. The majority of the code is devoted to holiday definitions.

Exhibit F

ALL-STATE® LEGAL 800-222-0510 EDR11 RECYCLED

Exhibit G

ALL-STATE® LEGAL 800-222-0510 EDR11 RECYCLED

```
!               -- Century Conversion --
!
!               Bruce Dickens   Apr 04, 1996 *
!
10   open structure tools:name 'otms_src_dir:tools'
     open #2 : name 'last_inv.dat', access output
        print "    Tools 'Last Inventory Data Format' Check for 1996 Inventory"
        print "ToolNo                   "; "  Model No  "; "  LAST_INV "; "LAST_INV "
        print "======                   "; "  ========  "; "  ======== "; "======== "
        print "Extract Data:"
        print #2: "ToolNo                   "; "  Model No  "; "  LAST_INV "; "LAST_I
NV "
        print #2: "======                   "; "  ========  "; "  ======== "; "======
== "
        print #2: "Extract Data:"

20   extract structure tools
        yy$ = lpad$ (element$(tools(last_inv),3,"/"), 2, "0" )
        mm$ = lpad$ (element$(tools(last_inv),1,"/"), 2, "0" )
        dd$ = lpad$ (element$(tools(last_inv),2,"/"), 2, "0" )
        cc$= yy$ + "/" +  mm$ + "/" + dd$
        c1$ = change$(cc$,'/','')
        if c1$[1:2] < '50' then
          c$ = '20' + c1$
        else
          c$= '19' + c1$
        end if
!       include c$ < '19960101'
          sort by tools(model)
          sort by rpad$(c$,8, '0')
!      if c$[1:8] < '19960101'  then
        print tools(toolno); tab(23); tools(model);   &
                           tab(35);tools(last_inv); tab(44); c$
        print #2: tools(toolno); tab(23); tools(model);   &
                           tab(35);tools(last_inv); tab(44); c$
                if valid ( c1$, "digits" ) = 0 then
                print; tab(53); " Date format is not digits"
                print #2:; tab(53); " Date format is not digits"
                end if
!               if valid ( c1$, "minlength 6" ) = 0 then
!               print ;tab(50); " Date format is short"
!               print #2: ;tab(50); " Date format is short"
!               end if
                if tools(last_inv) = "" then
                print ;tab(53); " Date format is blank "
                print #2: ;tab(53); " Date format is blank "
                end if
!      end if
30   end extract
        print
        print "Sorted Data:"
        print
40   for each tools
        c1$ = change$(tools(last_inv),'/','')
        print tools(toolno); tab(23); tools(model); &
         tab(35); tools(last_inv); tab(44); c$
        print #2: tools(toolno); tab(23); tools(model); &
         tab(35); tools(last_inv); tab(44); c$
                if valid ( c1$, "digits" ) = 0 then
                print ;tab(53); " Date format is not digits"
                print #2: ;tab(53); " Date format is not digits"
                end if
!               if valid ( c1$, "minlength 6" ) = 0 then
!               print ;tab(53); " Date format is short"
!               print #2: ;tab(53); " Date format is short"
!               end if
```

Attorney Docket No. 08190-0119.000 PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Dickens
Serial No.: 08/725,574
Filed: October 3, 1996
For: DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

Group Art Unit: 2771

Examiner: W. Amsbury

OIPE JC21 MAR 20 1998 PATENT & TRADEMARK OFFICE

March 17, 1998

Assistant Commissioner for Patents
Washington, DC 20231

SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT UNDER 37 C.F.R. § 1.97

Sir:

Attached is form PTO-1449 listing one additional document together with a copy of the identified document. This Information Disclosure Statement is submitted in accordance with 37 C.F.R. § 1.97(c), before final Office Action or Allowance, whichever is earlier.

In accordance with the requirements of 37 C.F.R. § 1.97(c), the following Certification as specified in 37 C.F.R. § 1.97(e) is made:

No item of information contained in this Statement was cited from a foreign patent office in a counterpart foreign application, or to the knowledge of the person signing this document after making reasonable inquiry, was known to any individual designated in 37 C.F.R. § 1.56(c) more than three (3) months prior to the filing of this Statement.

Respectfully submitted,

Guy R. Gosnell
Registration No. 34,610

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, NC 28234
Telephone (704) 331-6000
Facsimile (704) 334-2014

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Assistant Commissioner for Patents, Washington, DC 20231, on March 17, 1998.

W. Kevin Ransom

FORM PTO-1449 U.S. Department of Commerce Patent and Trademark Office LIST OF DOCUMENTS CITED BY APPLICANT (Use several sheets if necessary)	Attorney Docket Number 08190-0119.000	Serial Number 08/725,574
	Applicant: Dickens	
	Filing Date: October 3, 1996	Group 2771

OIPE JC21 MAR 20 1998 PATENT & TRADEMARK OFFICE

U. S. PATENT DOCUMENTS

Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date if Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

FOREIGN PATENT DOCUMENTS

		Document Number	Date	Country	Class	Subclass	Translation Yes | No
	L						
	M						
	N						
	O						
	P						

OTHER DOCUMENTS (Including Author, Title, Date, Pertinent Pages, Etc.)

[initials]	Q	IBM: *The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation*; First Edition, October 1995
	R	
	S	

EXAMINER Amsbury	DATE CONSIDERED 4/1/98

*EXAMINER Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. 313945

OFFICIAL

PATENT

#9/A

Attorney's Docket No. 08190-0119.000

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Bruce Dickens
Serial No.: 08/725,574
Filed: October 3, 1996
For: DATE FORMATTING AND SORTING FOR SPANNING THE TURN OF THE CENTURY

Group Art Unit: 2307
Examiner: W. Amsbury

April 2, 1998

Assistant Commissioner for Patents
Washington, DC 20231

SUPPLEMENTAL RESPONSE

Sir:

This Supplemental Response is intended to supplement the Response filed March 17, 1998 by amending the above-identified patent application as follows:

In The Claims:

Please amend independent Claims 1, 4, 6, 8, 10, 11 and 13 as follows:

1. (Amended) A method of processing symbolic representations of dates stored in a database, comprising the steps of

providing a database with symbolic representations of dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later

than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each symbolic representation of a date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B; and

reformatting the symbolic representation of the date [in the database] with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2 to facilitate further processing of the dates.

Claim 4, line 3, please delete "in the database".

Claim 6, line 3, please delete "in the database".

Claim 8, line 1, please delete "1", and insert -- 10 -- therefor.

Claim 10, line 3, please delete "sorted".

11. (Amended) A method of processing dates in a database, comprising the steps of

providing a database with dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of dates falling within a 10-decade period of

time which includes the decade beginning in the year 2000;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B; [and]

reformatting each date [in the database] in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$ to facilitate further processing of the dates; and

sorting the dates [in the database] in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$.

Claim 13, line 1, please delete "11", and insert -- 15 -- therefor.

REMARKS

As noted by the prior Response filed March 17, 1998, the first Official Action objected to the disclosure for implying that the current invention does not require additional data fields for storage to solve the year 2000 problem. The Official Action also rejected all of the claims under 35 U.S.C. § 112, first paragraph, as based on a disclosure that is not enabling. In particular, the Official Action stated that the conversion of the existing symbolic

date representations without the addition of new data fields is critical or essential to the invention, but is not included in the claims.

As stated in the Background Of The Invention section of the present application, conventional date formatting systems typically require additional data fields for storage to accommodate the century designators. These additional data fields are necessary because conventional systems permanently reformat the stored data. In contrast, the method of the present invention, as described by the specification, does not require that the converted or reformatted data that includes century designations be stored in data storage. Accordingly, independent Claims 1 and 11 have now been amended so as to not require storage of the converted dates, thereby not imposing any requirement for new data fields.

Thus, the method of the claimed invention encompasses embodiments in which the date information is initially reformatted and converted to have century designations, but does not require that the reformatted dates be stored. For example, the method of one embodiment of the claimed invention reads the dates from the database and temporarily reformats the dates with century designations. Data manipulation programs are then performed on these reformatted dates, such as sorting the dates. However, once the data manipulations are complete, the reformatted dates need not be stored in data storage. Instead,

the dates in data storage can remain the same as they were prior to the temporary reformatting of the date information. Thus, the method of this embodiment of the claimed invention does not require additional data fields for storage because the reformatted dates with century designations are only used "on the fly" for data manipulation and are not stored in data storage.

In view of the amendments to the independent claims, dependent Claims 4 and 6 have also been amended to no longer refer to symbolic representations of dates "in the database". Since dependent Claims 10 and 15 further recite storing the reformatted dates in the database, dependent Claims 8 and 13 which further describe manipulating information in the database that includes the reformatted dates have also been amended to depend from dependent Claims 10 and 15, respectively. Finally, dependent Claim 10 has been amended to delete reference to "sorted" symbolic representations of the dates to provide proper antecedent basis.

For the reasons described above, Applicant therefore submits that the disclosure is enabling and that both the disclosure and the claims, as amended, comply with 35 U.S.C. § 112, first paragraph. As such, the rejections raised by the first Official Action under 35 U.S.C. § 112 are therefore overcome.

CONCLUSION

In view of the amended claims and the remarks presented above, it is respectively submitted that all of the present claims of the application are in condition for immediate allowance. It is therefore respectfully requested that a Notice of Allowance be issued. The Examiner is encouraged to contact Applicant's undersigned attorney to resolve any remaining issues in order to expedite examination of the present application. The Commissioner is also hereby authorized to charge our Deposit Account No. 16-0605 for any additional extensions of time that are required for entry and consideration of this Supplemental Response beyond the one month extension of time that was previously obtained in conjunction with the prior Response.

Respectfully submitted,

Guy R. Gosnell
Registration No. 34,610

BELL SELTZER INTELLECTUAL PROPERTY LAW GROUP
ALSTON & BIRD LLP
Post Office Drawer 34009
Charlotte, NC 28234
Tel (704) 331-6000
Fax (704) 334-2014

CERTIFICATION OF FACSIMILE TRANSMISSION

I hereby certify that this paper is being facsimile transmitted to the Patent and Trademark Office on the date shown below.

Gwen Frickhoeffer — April 2, 1998
Date

316124

BELL SELTZER
Intellectual Property Law Group
ALSTON&BIRD LLP

1211 East Morehead Street
P.O. Drawer 34009
Charlotte, NC 28234-4009
704-331-6000
Fax: 704-334-2014

Paper # 9

OFFICIAL

TELECOPY

PLEASE DELIVER THE FOLLOWING MATERIAL AS SOON AS POSSIBLE

The information contained in this facsimile message and its contents are legally privileged and confidential information intended solely for the use of the addressee. If the reader of this message is not the intended recipient, you are hereby notified that any dissemination, distribution, copying or other use of this message and its contents is strictly prohibited. If you have received this telecopy in error, please notify us immediately by telephone and return the original message to us at the address shown above via the United States Postal Service. Thank You.

DATE:	April 2, 1998
TO:	**Examiner Wayne Amsbury, Group Art Unit 2307**
	U.S. Patent And Trademark Office
RE:	**Bruce Dickens, Serial No. 08/725,574, filed October 3, 1996 For: DATE FORMATTING AND SORTING FOR SPANNING THE TURN OF THE CENTURY.**
FROM:	Guy R. Gosnell, Esq.

MESSAGE:

Following is the Supplemental Response we have discussed.

NO. OF PAGES: 7 (Including cover page)	**FAX OPERATOR:**
USER CODE:	FAX NUMBER: (703) 305 9731
CLIENT/MATTER: 08190-0119.000	INTERNATIONAL: (011)
REQUESTED BY: Gwen Ext: 6173	ADDRESSEE TELEPHONE NO.: (703) 305 3828

IF NOT RECEIVED PROPERLY, PLEASE NOTIFY US IMMEDIATELY AT (704) 331-6115

310141

A

#10

Notice of Allowability	Application No. 08/725,574	Applicant(s) Dickens
	Examiner Wayne Amsbury	Group Art Unit 2771

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance and Issue Fee Due or other appropriate communication will be mailed in due course.

☒ This communication is responsive to *faxed supplementary amendment of 4/2/98*.

☒ The allowed claim(s) is/are *1-15*.

☐ The drawings filed on ________ are acceptable.

☐ Acknowledgement is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d).

☐ All ☐ Some* ☐ None of the CERTIFIED copies of the priority documents have been

☐ received.

☐ received in Application No. (Series Code/Serial Number) ________.

☐ received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

*Certified copies not received: ________.

☐ Acknowledgement is made of a claim for domestic priority under 35 U.S.C. § 119(e).

A SHORTENED STATUTORY PERIOD FOR RESPONSE to comply with the requirements noted below is set to EXPIRE **THREE MONTHS** FROM THE "DATE MAILED" of this Office action. Failure to timely comply will result in ABANDONMENT of this application. Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

☐ Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL APPLICATION, PTO-152, which discloses that the oath or declaration is deficient. A SUBSTITUTE OATH OR DECLARATION IS REQUIRED.

☒ Applicant MUST submit NEW FORMAL DRAWINGS

☒ because the originally filed drawings were declared by applicant to be informal.

☒ including changes required by the Notice of Draftsperson's Patent Drawing Review, PTO-948, attached hereto or to Paper No. 5.

☐ including changes required by the proposed drawing correction filed on ________, which has been approved by the examiner.

☒ including changes required by the attached Examiner's Amendment/Comment.

Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the reverse side of the drawings. The drawings should be filed as a separate paper with a transmittal lettter addressed to the Official Draftsperson.

☐ Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Any response to this letter should include, in the upper right hand corner, the APPLICATION NUMBER (SERIES CODE/SERIAL NUMBER). If applicant has received a Notice of Allowance and Issue Fee Due, the ISSUE BATCH NUMBER and DATE of the NOTICE OF ALLOWANCE should also be included.

Attachment(s)

☐ Notice of References Cited, PTO-892

☐ Information Disclosure Statement(s), PTO-1449, Paper No(s). ________

☐ Notice of Draftsperson's Patent Drawing Review, PTO-948

☐ Notice of Informal Patent Application, PTO-152

☒ Interview Summary, PTO-413

☒ Examiner's Amendment/Comment

☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material

☒ Examiner's Statement of Reasons for Allowance

CLAIMS 1-15 ARE PENDING

1. The drawings are objected to on the grounds that FIG 2 does not conform to the claims as amended. In particular, is shows box **36** labeled: reformat data in database. The summary of the invention, the statements at the bottom of page 3 of the Response, and the Reasons for Allowance below specifically preclude this step. This box should be removed.

2. The following is an examiner's statement of reasons for allowance:

The Prior Art of Record, taking into account the Affidavit of the inventor, received 3/24/98, swearing behind the reference of the previous action, does not anticipate nor suggest the set of limitations of the claims, comprising the threshold year digits as used to determine a pair of century digits to be used for computation, but without enlarging the number of date digits of the database.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

3. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Wayne Amsbury whose telephone number is (703) 305-3828. The examiner can normally be reached on Monday-Thursday from 6:30 AM to 5:00 PM Eastern time.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas G. Black, can be reached on (703) 305-9707. The fax phone number for this Art Unit is (703) 305-9731.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-9600.

WAYNE AMSBURY
PRIMARY PATENT EXAMINER

April 2, 1998

Interview Summary	Application No. 08/725,574	Applicant(s) Dickens
	Examiner Wayne Amsbury	Group Art Unit 2771

All participants (applicant, applicant's representative, PTO personnel):

(1) *Wayne Amsbury* (3) ______

(2) *Guy Gosnell* (4) ______

Date of Interview *Apr 2, 1998*

Type: ☒ Telephonic ☐ Personal (copy is given to ☐ applicant ☐ applicant's representative).

Exhibit shown or demonstration conducted: ☐ Yes ☒ No. If yes, brief description:

Agreement ☒ was reached. ☐ was not reached.

Claim(s) discussed: *1-15*

Identification of prior art discussed:
Art of record in the case.

Description of the general nature of what was agreed to if an agreement was reached, or any other comments:
It was agreed that the summary of the invention, and the arguments of the response, were not entirely in conformity with the claims, which would be potentially allowable if the use of additional century digits did not include their storage in the database. It was further agreed that Applicant would fax a supplementary amendment to address this problem.

(A fuller description, if necessary, and a copy of the amendments, if available, which the examiner agreed would render the claims allowable must be attached. Also, where no copy of the amendents which would render the claims allowable is available, a summary thereof must be attached.)

1. ☒ It is not necessary for applicant to provide a separate record of the substance of the interview.

Unless the paragraph above has been checked to indicate to the contrary, A FORMAL WRITTEN RESPONSE TO THE LAST OFFICE ACTION IS NOT WAIVED AND MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW. (See MPEP Section 713.04). If a response to the last Office action has already been filed, APPLICANT IS GIVEN ONE MONTH FROM THIS INTERVIEW DATE TO FILE A STATEMENT OF THE SUBSTANCE OF THE INTERVIEW.

2. ☐ Since the Examiner's interview summary above (including any attachments) reflects a complete response to each of the objections, rejections and requirements that may be present in the last Office action, and since the claims are now allowable, this completed form is considered to fulfill the response requirements of the last Office action. Applicant is not relieved from providing a separate record of the interview unless box 1 above is also checked.

WAYNE AMSBURY
PRIMARY PATENT EXAMINER

Examiner Note: You must sign and stamp this form unless it is an attachment to a signed Office action.

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office

NOTICE OF ALLOWANCE AND ISSUE FEE DUE

LM41/0408

GUY R. GOSNELL
BELL, SELTZER, PARK & GIBSON, P.A.
P.O. DRAWER 34009
CHARLOTTE NC 28234

APPLICATION NO.	FILING DATE	TOTAL CLAIMS	EXAMINER AND GROUP ART UNIT		DATE MAILED
08/725,574	10/03/96	015	AMSBURY, W	2771	04/08/98

First Named Applicant: DICKENS, BRUCE

TITLE OF INVENTION: DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

ATTY'S DOCKET NO.	CLASS-SUBCLASS	BATCH NO.	APPLN. TYPE	SMALL ENTITY	FEE DUE	DATE DUE
2 11151	707-006.000	C70	UTILITY	NO	$1320.00	07/08/98

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED.

THE ISSUE FEE MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED.

HOW TO RESPOND TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is changed, pay twice the amount of the FEE DUE shown above and notify the Patent and Trademark Office of the change in status, or

B. If the status is the same, pay the FEE DUE shown above.

If the SMALL ENTITY is shown as NO:

A. Pay FEE DUE shown above, or

B. File verified statement of Small Entity Status before, or with, payment of 1/2 the FEE DUE shown above.

II. Part B-Issue Fee Transmittal should be completed and returned to the Patent and Trademark Office (PTO) with your ISSUE FEE. Even if the ISSUE FEE has already been paid by charge to deposit account, Part B Issue Fee Transmittal should be completed and returned. If you are charging the ISSUE FEE to your deposit account, section "4b" of Part B-Issue Fee Transmittal should be completed and an extra copy of the form should be submitted.

III. All communications regarding this application must give application number and batch number. Please direct all communications prior to issuance to Box ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PTOL-85 (REV. 10-96) Approved for use through 06/30/99. (0651-0033)

PATENT AND TRADEMARK OFFICE COPY

☆ U.S. GOVERNMENT PRINTING OFFICE 1997-433-221/82718

410

707/006

1

Attorney's Docket No. 08190-0119.000 PATENT

OIPE
MAY 11 1998
C38
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

E

In re: Dickens
Serial No.: 08/725,574
Filed: 10/03/96
For: DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

Date of Notice of Allowance: 04/08/98
Issue Batch No.: C70
May 7, 1998

#11 OK

Drawing Review Branch
Assistant Commissioner for Patents
Washington, DC 20231

SUBMITTAL OF FORMAL DRAWINGS

Sir:

In response to the requirement for new drawings as set forth in Notice of Allowability in the above application, there is enclosed herewith one set (1 sheet) of new formal drawings. It is requested that these new drawings be substituted for the originally filed informal drawings.

Respectfully submitted,

Guy R. Gosnell
Registration No. 34,610

BELL SELTZER INTELLECTUAL PROPERTY LAW GROUP
ALSTON & BIRD LLP
Post Office Drawer 34009
Charlotte, NC 28234
Tel (704) 331-6000
Fax (704) 334-2014

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to:
Drawing Review Branch, Assistant Commissioner for Patents, Washington, DC 20231, on May 7, 1998.

Elizabeth Herbener

319053

HJ

141-1520
561-30

PART B—ISSUE FEE TRANSMITTAL

Complete and mail this form, together with applicable fees, to: **Box ISSUE FEE**
Assistant Commissioner for Patents
Washington, D.C. 20231

MAILING INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE. Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Issue Fee Receipt, the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use Block 1)

[illegible]

MAY 22 1998

Note: The certificate of mailing below can only be used for domestic mailings of the Issue Fee Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing.

Certificate of Mailing

I hereby certify that this Issue Fee Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Box Issue Fee address above on the date indicated below.

Elizabeth Herbener (Depositor's name)

Elizabeth Herbener (Signature)

5-19-98 (Date)

APPLICATION NO.	FILING DATE	TOTAL CLAIMS	EXAMINER AND GROUP ART UNIT	DATE MAILED
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

First Named Applicant [illegible]

TITLE OF INVENTION [illegible]

ATTY'S DOCKET NO.	CLASS-SUBCLASS	BATCH NO.	APPLN. TYPE	SMALL ENTITY	FEE DUE	DATE DUE
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363). Use of PTO form(s) and Customer Number are recommended, but not required.

☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47) attached.

2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 Bell Seltzer Intellectual Property Group of Alston & Bird LLP
2 ______
3 ______

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)
PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropiate when an assignment has been previously submitted to the PTO or is being submitted under separate cover. Completion of this form is NOT a subsititue for filing an assignment.

(A) NAME OF ASSIGNEE McDonnell Douglas Corporation

(B) RESIDENCE: (CITY & STATE OR COUNTRY) Long Beach, CA

Please check the appropriate assignee category indicated below (will not be printed on the patent)

☐ individual ☐ corporation or other private group entity ☐ government

4a. The following fees are enclosed (make check payable to Commissioner of Patents and Trademarks):

☒ Issue Fee
☒ Advance Order - # of Copies 10

4b. The following ~~fees~~ or deficiency in these fees should be charged to:
DEPOSIT ACCOUNT NUMBER 16-0605
(ENCLOSE AN EXTRA COPY OF THIS FORM)

☐ Issue Fee
☐ Advance Order - # of Copies ______

The COMMISSIONER OF PATENTS AND TRADEMARKS IS requested to apply the Issue Fee to the application identified above.

(Authorized Signature) [signature] (Date) 5/19/98

NOTE; The Issue Fee will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the Patent and Trademark Office.

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending on the needs of the individual case. Any comments on the amount of time required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, D.C. 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND FEES AND THIS FORM TO: Box Issue Fee, Assistant Commissioner for Patents, Washington D.C. 20231

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

05/27/1998 CASHBY 00000018 08725574

01 FC:142 1320.00 OP
02 FC:561 30.00 OP

TRANSMIT THIS FORM WITH FEE

APPROVED BY DRAFTSMAN	O.G. FIG.	
	CLASS	SUBCLASS

5806063

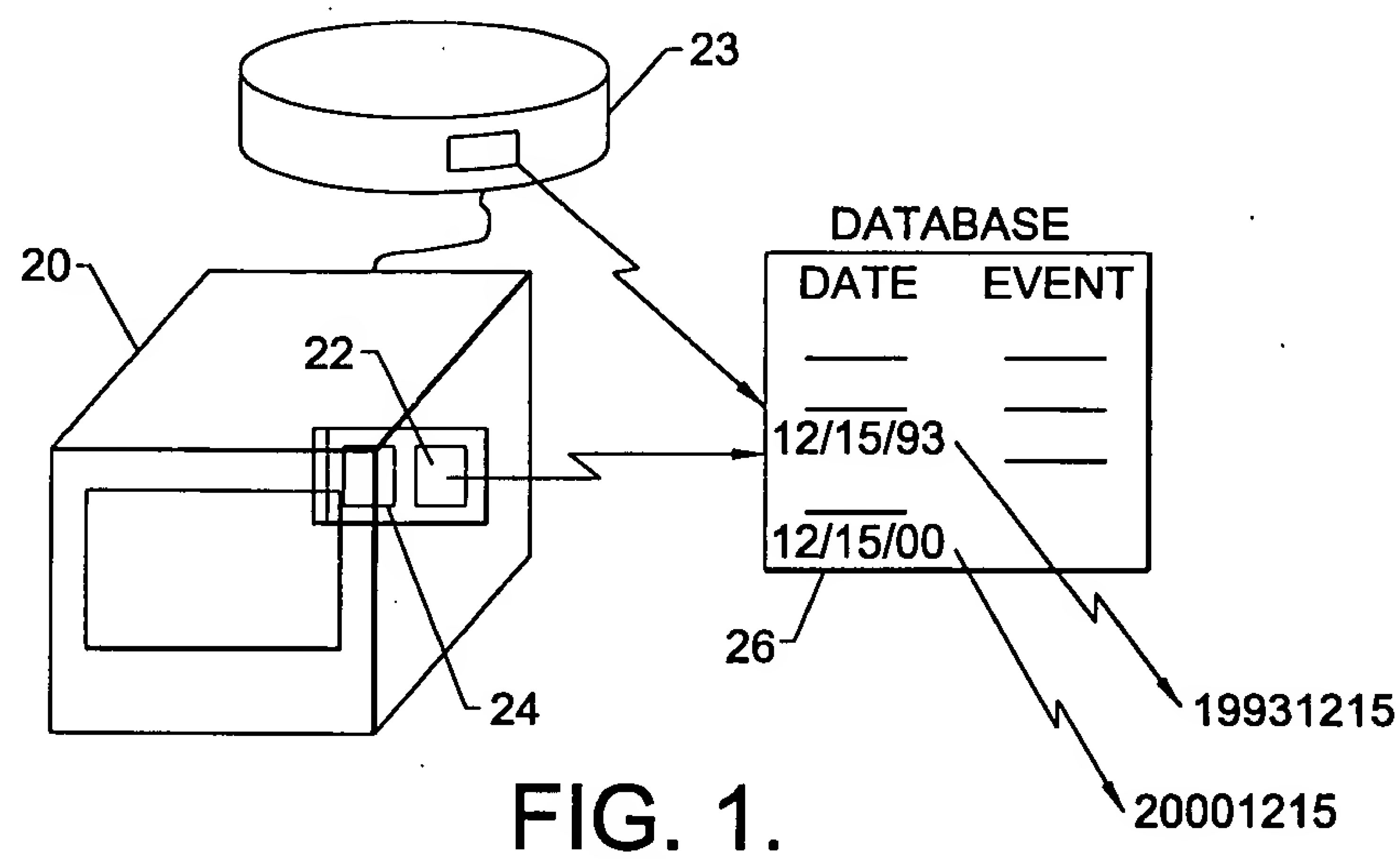


FIG. 1.

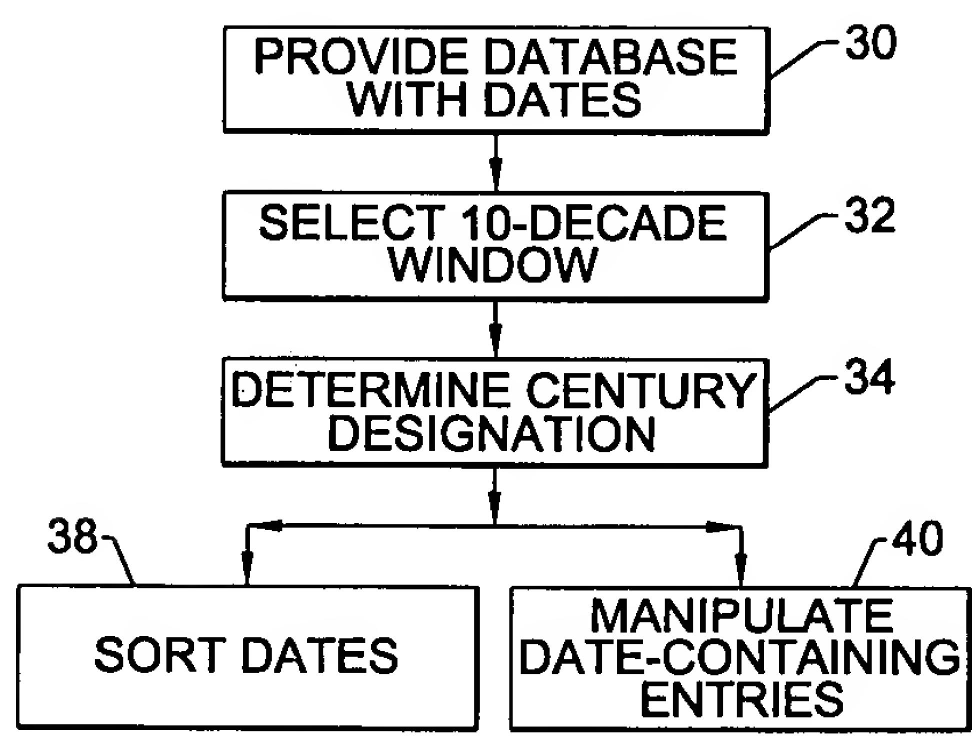


FIG. 2.

Serial No: 08/725,574
Inventor(s): Bruce Dickens
Atty Dkt: 08190.0119.000
Attorney: Guy R. Gosnell, Esq.
Tel. No.: (704)331-6000
Page 1 of 1

O I P E
MAY 11 1998
PATENT & TRADEMARK OFFICE
JC38

BELL SELTZER
Intellectual Property Law Group
ALSTON&BIRD LLP

1211 East Morehead Street
P.O. Drawer 34009
Charlotte, North Carolina 28234-4009

704-331-6000
Fax: 704-334-2014

#13
mjb

October 5, 1998

NOV 30 1998

Assistant Commissioner for Patents
Washington, DC 20231

Re: United States Patent for "Date Formatting And Sorting For Dates Spanning The Turn Of The Century"
Application No. 08/725,574
Filed October 3, 1996
Patent No. 5,806,063
Issued September 8, 1998
Our File 8190-119 (038190/160265)

Sir:

It is respectfully requested that a Certificate of Correction be issued for the above-identified patent, in accordance with 37 C.F.R. § 1.322. This request is made in order to correct the mistakes incurred through the fault of the Patent Office.

The mistakes appearing in the patent are set forth on the Certificate of Correction enclosed herewith, with an additional copy thereof and a postal card being enclosed in accordance with the present Patent Office practice.

Respectfully submitted,

Guy R. Gosnell
Registration No. 34,610

GRG/mhf:338686
Enclosures

Staple Here Only !

PRINTER'S TRIM LINE

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,806,063

DATED : September 8, 1998

INVENTOR(S) : Dickens

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the References Cited, OTHER PUBLICATIONS, line 1, before "The", insert --IBM: --.

In the ABSTRACT, line 4, "$_{M1}M_2$" should read --M_1M_2--.

MAILING ADDRESS OF SENDER:

BELL SELTZER INTELLECTUAL PROPERTY LAW GROUP
ALSTON & BIRD LLP
Post Office Drawer 34009
Charlotte, NC 28234

PATENT NO. 5,806,063

No. of add'l. copies @ 30¢ per page

FORM PTO 1050 (REV. 3-82)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,806,063

DATED : September 8, 1998

INVENTOR(S) : Dickens

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On title page, item [56],

In the References Cited, OTHER PUBLICATIONS, line 1, before "The", insert --IBM: --.

In the ABSTRACT, line 4, "${}_{M1}M_2$" should read --M_1M_2--.

Signed and Sealed this

Twenty-ninth Day of December, 1998

Attest:

Bruce Lehman

BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

BAR CODE LABEL

U.S. PATENT APPLICATION

SERIAL NUMBER	FILING DATE	CLASS	GROUP ART UNIT
08/725,574	10/03/96	395	2309

APPLICANT

BRUCE DICKENS, IRVINE, CA.

CONTINUING DATA********************
VERIFIED

FOREIGN/PCT APPLICATIONS**********
VERIFIED

FOREIGN FILING LICENSE GRANTED 12/07/96

STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS	INDEPENDENT CLAIMS	FILING FEE RECEIVED	ATTORNEY DOCKET NO.
CA	1	15	2	$770.00	11151

ADDRESS

GREGORY O GARMONG
PO BOX 12460
ZEPHYR COVE NV 89448

TITLE

DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

This is to certify that annexed hereto is a true copy from the records of the United States Patent and Trademark Office of the application which is identified above.

By authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

Date

Certifying Officer

MULTIPLE DEPENDENT CLAIM
FEE CALCULATION SHEET
(FOR USE WITH FORM PTO-875)

SERIAL NO. 725574

FILING DATE

APPLICANT(S)

CLAIMS

	AS FILED		AFTER 1st AMENDMENT		AFTER 2nd AMENDMENT	
	IND.	DEP.	IND.	DEP.	IND.	DEP.
1	1					
2		1				
3		1				
4		1				
5		1				
6		1				
7		1				
8		1				
9		1				
10		1				
11	1					
12		1				
13		1				
14		1				
15		1				
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						
33						
34						
35						
36						
37						
38						
39						
40						
41						
42						
43						
44						
45						
46						
47						
48						
49						
50						
TOTAL IND.	2					
TOTAL DEP.	13					
TOTAL	15					

	•		•		•	
	IND.	DEP.	IND.	DEP.	IND.	DEP.
51						
52						
53						
54						
55						
56						
57						
58						
59						
60						
61						
62						
63						
64						
65						
66						
67						
68						
69						
70						
71						
72						
73						
74						
75						
76						
77						
78						
79						
80						
81						
82						
83						
84						
85						
86						
87						
88						
89						
90						
91						
92						
93						
94						
95						
96						
97						
98						
99						
100						
TOTAL IND.						
TOTAL DEP.						
TOTAL						

PATENT APPLICATION FEE DETERMINATION RECORD

Effective October 1, 1996

Application or Docket Number
725574

CLAIMS AS FILED - PART I

FOR	NUMBER FILED (Column 1)	NUMBER EXTRA (Column 2)	SMALL ENTITY RATE	SMALL ENTITY FEE	OR	OTHER THAN SMALL ENTITY RATE	OTHER THAN SMALL ENTITY FEE
BASIC FEE				385.00	OR		770.00
TOTAL CLAIMS	15 minus 20 =	*	x$11=		OR	x$22=	
INDEPENDENT CLAIMS	2 minus 3 =	*	x40=		OR	x80=	
MULTIPLE DEPENDENT CLAIM PRESENT			+130=		OR	+260=	
			TOTAL		OR	TOTAL	770

* If the difference in column 1 is less than zero, enter "0" in column 2

CLAIMS AS AMENDED - PART II

AMENDMENT A

	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	SMALL ENTITY RATE	ADDITIONAL FEE	OR	OTHER THAN SMALL ENTITY RATE	ADDITIONAL FEE
Total	*	Minus	**	=	x$11=		OR	x$22=	
Independent	*	Minus	***	=	x40=		OR	x80=	
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
					TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	RATE	ADDITIONAL FEE	OR	RATE	ADDITIONAL FEE
Total	*	Minus	**	=	x$11=		OR	x$22=	
Independent	*	Minus	***	=	x40=		OR	x80=	
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
					TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	RATE	ADDITIONAL FEE	OR	RATE	ADDITIONAL FEE
Total	*	Minus	**	=	x$11=		OR	x$22=	
Independent	*	Minus	***	=	x40=		OR	x80=	
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
					TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Form PTO 1130
(REV 2/94)

U.S. DEPARTMENT OF COMMERCE
Patent and Trademark Office

PACE DATA ENTRY CODING SHEET

1ST EXAMINER	B. Sonleo	DATE	12/6
2ND EXAMINER		DATE	

APPLICATION NUMBER	TYPE APPL	FILING DATE (MONTH DAY YEAR)	SPECIAL HANDLING	GROUP ART UNIT	CLASS	SHEETS OF DRAWING
08 725574	1	10 03 96	0	2309	395	1

TOTAL CLAIMS	INDEPENDENT CLAIMS	SMALL ENTITY?	FILING FEE	FOREIGN LICENSE	ATTORNEY DOCKET NUMBER
15	2	0	770	Y	11151

CONTINUITY DATA

CONT CODE	STATUS CODE	PARENT APPLICATION SERIAL NUMBER	PCT APPLICATION SERIAL NUMBER	PARENT PATENT NUMBER	PARENT FILING DATE (MONTH DAY YEAR)
			PCT/ /		
			PCT/ /		
			PCT/ /		
			PCT/ /		
			PCT/ /		

PCT/FOREIGN APPLICATION DATA

FOREIGN PRIORITY CLAIMED	COUNTRY CODE	PCT/FOREIGN APPLICATION SERIAL NUMBER	FOREIGN FILING DATE (MONTH DAY YEAR)

TITLE OF INVENTION

ATTORNEY REGISTRATION NUMBERS

CORRESPONDENCE NAME AND ADDRESS

APPLICANT/INVENTOR DATA

AUTHORITY CODE

FAMILY NAME

NAME SUFFIX

GIVEN NAME

STATE/CTRY CODE

CITY

AUTHORITY CODE

FAMILY NAME

NAME SUFFIX

GIVEN NAME

STATE/CTRY CODE

CITY

MORE

*U.S. Government Printing Office: 1996 - 404-474/40903

FORM PTO-447A
(Rev. 2-95)

Staple to Front of Application

U.S. DEPARTMENT OF COMME
PATENT & TRADEMARK OF

APPLICATION TRANSFER REQUEST

Section I. APPLICATION TRANSFER REQUEST Date 08/01/97 S.N. 725 574

TO: Receiving A.U. 2307 Class/sub 395/611 Examiner

FROM: Originating A.U. 2415 Class/Sub 395/326+ Examiner Y. NGUY[illegible]

REASON: ☐ Request for Reconsideration (Return to Classification)

Consider 395/611 for processing date format in a database for sorting

---

Section II. DISPOSITION BY RECEIVING A.U. Date ______ Ex'r ______

☐ **Accepted (keep in receiving A.U.)**

Not Accepted ☐ Forward to ______ Classification Group

☐ Return to Originating A.U. ______ Nonclassification issue only:

REASON:

☐ Restriction
☐ Other

Section III. DISPOSITION BY ______ Classification Group. Date 9/1/97

☑ Transfer Approved-Forward to A.U. 2307 Class/sub 395/611 Classifier J. MILLS

☐ Transfer Disapproved-Forward to Originating A.U. ______ Concurring Classifier ______

REASON:

Nonclassification issue raised: ☐ Restriction ☐ Other

★ U.S. GPO: 1996-

D HIS

```
     (FILE 'USPAT' ENTERED AT 15:16:28 ON 02 APR 1998)
L1           326 S Y2K OR (YEAR 2000)
L2            25 S L1 AND CENTURY
L3         18551 S DATABASE OR (DATA BASE)
L4             3 S L2 AND L3
L5            20 S L1 AND L3
L6            28 S L1 AND WINDOW
L7            21 S L6 NOT L5
```

=> D 1-21

1. 5,709,734, Jan. 20, 1998, Method for disposing of ozone-degrading and climatically active halogenated compounds; Christoph Scholz, et al., 95/131; 588/206 [IMAGE AVAILABLE]

2. 5,656,821, Aug. 12, 1997, Quantum semiconductor device with triangular etch pit; Yoshiki Sakuma, 257/14, 15, 17, 22, 627; 372/45 [IMAGE AVAILABLE]

3. 5,552,017, Sep. 3, 1996, Method for improving the process uniformity in a reactor by asymmetrically adjusting the reactant gas flow; Syun-Ming Jang, et al., 438/710; 118/715, 723E; 156/345; 427/248.1, 569; 438/935, 941 [IMAGE AVAILABLE]

4. 5,511,568, Apr. 30, 1996, Endoscopic cannulated instrument flushing apparatus for forcing a cleaning solution through an endoscopic cannulated instrument for removal of gross debris; Michael D. Bowman, et al., 134/102.2, 166C, 169C; 222/401 [IMAGE AVAILABLE]

5. 5,499,303, Mar. 12, 1996, Correction of the gaze direction for a videophone; Eckart Hundt, et al., 382/100; 348/78; 382/173, 293 [IMAGE AVAILABLE]

6. 5,495,238, Feb. 27, 1996, Induction watt-hour meter non-intrusive and concealed pulse initiator; Steven A. Baker, et al., 340/870.02; 324/74, 96 [IMAGE AVAILABLE]

7. 5,425,426, Jun. 20, 1995, Fire extinguishing methods and systems; Anatoly Baratov, et al., 169/46, 12 [IMAGE AVAILABLE]

8. 5,423,385, Jun. 13, 1995, Fire extinguishing methods and systems; Anatoly Baratov, et al., 169/46, 12, 66 [IMAGE AVAILABLE]

9. 5,363,096, Nov. 8, 1994, Method and apparatus for encoding-decoding a digital signal; Pierre Duhamel, et al., 341/50, 51 [IMAGE AVAILABLE]

10. 5,279,317, Jan. 18, 1994, Endoscopic cannulated instrument flushing apparatus for forcing a cleaning solution through an endoscopic cannulated instrument for removal of gross debris; Michael D. Bowman, et al., 134/166C, 169C, 170, 201 [IMAGE AVAILABLE]

11. 5,270,787, Dec. 14, 1993, Electro-optical methods and apparatus for high speed, multivariate measurement of individual entities in fiber or other samples; Frederick M. Shofner, et al., 356/238; 73/160; 356/383, 385 [IMAGE AVAILABLE]

12. 5,213,021, May 2. 1993, Reciprocating cutter ass ly; Billy D. Goforth, et al., 83/318, 578, 580 [IMAGE AVAILABLE]

13. 5,096,046, Mar. 17, 1992, System and process for making synthetic wood products from recycled materials; Billy D. Goforth, et al., 198/604; 193/35R; 198/620, 782 [IMAGE AVAILABLE]

14. 5,088,910, Feb. 18, 1992, System for making synthetic wood products from recycled materials; Billy D. Goforth, et al., 425/142; 83/289; 264/118, 122, 148, 914, 920; 425/131.1, 202, 205, 308, 377, DIG.46 [IMAGE AVAILABLE]

15. 4,772,559, Sep. 20, 1988, Method of detecting the presence of bronchogenic carcinoma by analysis of expired lung air; George Preti, et al., 436/64; 95/82; 436/96, 111, 140, 161, 813, 900; 600/532, 543 [IMAGE AVAILABLE]

16. 4,582,590, Apr. 15, 1986, Solar heated oil shale pyrolysis process; Shaik A. Qader, 208/409; 48/197R, DIG.9; 208/407 [IMAGE AVAILABLE]

17. 4,481,382, Nov. 6, 1984, Programmable telephone system; Antony-Euclid C. Villa-Real, 455/556; 360/137; 369/6, 10, 25, 69; 379/68, 85, 88, 101.01, 110.01; 455/412, 418 [IMAGE AVAILABLE]

18. 4,337,754, Jul. 6, 1982, Solar reflector and heat storage device; Steven J. Conger, 126/618, 633, 674, 685, 701, 710; 359/596 [IMAGE AVAILABLE]

19. 4,179,610, Dec. 18, 1979, Apparatus to indicate bio-rhythm curves; James T. Chester, 235/89R; 40/107; 116/308, 309, 323; 235/70A, 85R [IMAGE AVAILABLE]

20. 3,805,430, Apr. 23, 1974, PERPETUAL CALENDAR; C. Louis Smader, 40/113; D19/25 [IMAGE AVAILABLE]

21. 3,765,111, Oct. 16, 1973, PERPETUAL CALENDAR; Lauren D. Spicer, 40/111; D19/25 [IMAGE AVAILABLE]

D HIS

```
     (FILE 'USPAT' ENTERED AT 15:16:28 ON 02 APR 1998)
L1            326 S Y2K OR (YEAR 2000)
L2             25 S L1 AND CENTURY
L3          18551 S DATABASE OR (DATA BASE)
L4              3 S L2 AND L3
L5             20 S L1 AND L3
```

=> D 1-20

1. 5,719,949, Feb. 17, 1998, Process and apparatus for cross-correlating digital imagery; Gregory T. Koeln, et al., 382/113; 348/144; 382/109 [IMAGE AVAILABLE]

2. 5,719,826, Feb. 17, 1998, Calendaring system; Michael D. Lips, 368/29; 364/705.08 [IMAGE AVAILABLE]

3. 5,704,044, Dec. 30, 1997, Computerized healthcare accounts receivable purchasing, collections, securitization and management system; Fred B. Tarter, et al., 705/4, 2 [IMAGE AVAILABLE]

4. 5,691,134, Nov. 25, 1997, Poliovirus specific primers and methods of detection utilizing the same; David R. Kilpatrick, 435/5; 536/23.72 [IMAGE AVAILABLE]

5. 5,668,989, Sep. 16, 1997, Two-digit hybrid radix year numbers for **year 2000** and beyond; Decao Mao, 707/101; 341/82, 83, 84, 85; 364/744, 771; 395/704, 705; 705/25 [IMAGE AVAILABLE]

6. 5,644,762, Jul. 1, 1997, Method and apparatus for recording and reading date data having coexisting formats; Thomas B. Soeder, 707/6 [IMAGE AVAILABLE]

7. 5,644,354, Jul. 1, 1997, Interactive video system; John R. Thompson, et al., 348/13; 340/825.34; 348/12; 380/10, 20, 23 [IMAGE AVAILABLE]

8. 5,618,043, Apr. 8, 1997, Game based on **data base** of characters of different geographic regions; John J. McGlew, 273/308; 463/1, 9 [IMAGE AVAILABLE]

9. 5,590,133, Dec. 31, 1996, Apparatuses and mobile stations for providing packet data communication in digital TDMA cellular systems; Lars Billstrom, et al., 370/349, 332, 337, 338, 403; 455/433 [IMAGE AVAILABLE]

10. 5,585,477, Dec. 17, 1996, Poliovirus specific primers; David R. Kilpatrick, 536/23.72, 24.33 [IMAGE AVAILABLE]

11. 5,551,073, Aug. 27, 1996, Authentication key entry in cellular radio system; Anthony J. Sammarco, 455/411; 340/825.34; 380/23 [IMAGE AVAILABLE]

12. 5,550,734, Aug. 27, 1996, Computerized healthcare accounts receivable purchasing collections securitization and management system; Fred B. Tarter, et al., 705/2 [IMAGE AVAILABLE]

13. 5,538,291, Jul. ? 1996, Anti-theft credit card; ''f Gustafson, 235/487, 493; 283/904 MAGE AVAILABLE]

14. 5,533,890, Jul. 9, 1996, Method and apparatus for control of fugitive VOC emissions; Mark R. Holst, et al., 431/5; 422/177; 431/7, 170 [IMAGE AVAILABLE]

15. 5,507,133, Apr. 16, 1996, Inoculant method and apparatus; Paul Singleton, et al., 53/474, 239, 435 [IMAGE AVAILABLE]

16. 5,442,683, Aug. 15, 1995, Directory structure for large scale telecommunications network allowing location of roaming mobile subscribers; Jacobus Hoogeveen, 455/403, 435, 445 [IMAGE AVAILABLE]

17. 5,396,227, Mar. 7, 1995, Electronic system and method for monitoring compliance with a protective order; Gary T. Carroll, et al., 340/825.36, 573, 825.54; 379/38 [IMAGE AVAILABLE]

18. 5,266,944, Nov. 30, 1993, Electronic system and method for monitoring abusers for compliance with a protective order; Gary T. Carroll, et al., 340/825.36, 573, 825.54; 379/38 [IMAGE AVAILABLE]

19. 5,220,501, Jun. 15, 1993, Method and system for remote delivery of retail banking services; Matthew P. Lawlor, et al., 380/24; 379/93.18; 380/29; 705/43; 902/24 [IMAGE AVAILABLE]

20. 4,661,811, Apr. 28, 1987, Video map display; Michael J. Gray, et al., 345/113; 340/995; 345/132 [IMAGE AVAILABLE]

```
D HIS

     (FILE 'USPAT' ENTERED AT 15:16:28 ON 02 APR 1998)
L1          326 S Y2K OR (YEAR 2000)
L2           25 S L1 AND CENTURY
L3        18551 S DATABASE OR (DATA BASE)
L4            3 S L2 AND L3

=> D 1-3
```

1. 5,719,826, Feb. 17, 1998, Calendaring system; Michael D. Lips, 368/29; 364/705.08 [IMAGE AVAILABLE]

2. 5,668,989, Sep. 16, 1997, Two-digit hybrid radix year numbers for **year 2000** and beyond; Decao Mao, 707/101; 341/82, 83, 84, 85; 364/744, 771; 395/704, 705; 705/25 [IMAGE AVAILABLE]

3. 5,618,043, Apr. 8, 1997, Game based on **data base** of characters of different geographic regions; John J. McGlew, 273/308; 463/1, 9 [IMAGE AVAILABLE]

```
D HIS

     (FILE 'USPAT' ENTERED AT 13:35:17 ON 06 NOV 1997)
L1             6 S Y2K OR (2000 PROBLEM)
L2          2141 S (EBCDIC OR ASCII) AND DATE#
L3          1705 S L2 AND FORMAT
L4           790 S L3 AND DIGIT#
L5           719 S L4 AND FIELD#
L6           126 S L5 AND 2000
L7            29 S L6 AND YEAR
L8           321 S DD AND MM AND YY
L9           122 S L8 AND FORMAT
L10           82 S L9 AND DIGIT#
L11            5 S L10 AND CENTURY
L12           51 S L2 AND L10
L13           46 S L12 AND FIELD
L14            6 S L13 AND 2000
L15            2 S L8 AND (YEAR 2000)

=> D 1-2
```

1. 5,668,989, Sep. 16, 1997, Two-digit hybrid radix year numbers for **year 2000** and beyond; Decao Mao, 707/101; 341/82, 83, 84, 85; 364/744, 771; 395/704, 705; 705/25 :IMAGE AVAILABLE:

2. 5,630,118, May 13, 1997, System and method for modifying and operating a computer system to perform date operations on date fields spanning centuries; Daniel P. Shaughnessy, 707/1; 395/684 :IMAGE

D HIS

```
     (FILE 'USPAT' ENTERED AT 13:35:17 ON 06 NOV 1997)
L1            6 S Y2K OR (2000 PROBLEM)
L2         2141 S (EBCDIC OR ASCII) AND DATE#
L3         1705 S L2 AND FORMAT
L4          790 S L3 AND DIGIT#
L5          719 S L4 AND FIELD#
L6          126 S L5 AND 2000
L7           29 S L6 AND YEAR
L8          321 S DD AND MM AND YY
L9          122 S L8 AND FORMAT
L10          82 S L9 AND DIGIT#
L11           5 S L10 AND CENTURY
```

=> D 1-5

1. 5,668,989, Sep. 16, 1997, Two-**digit** hybrid radix year numbers for year 2000 and beyond; Decao Mao, 707/101; 341/82, 83, 84, 85; 364/744, 771; 395/704, 705; 705/25 :IMAGE AVAILABLE:

2. 5,630,118, May 13, 1997, System and method for modifying and operating a computer system to perform date operations on date fields spanning centuries; Daniel P. Shaughnessy, 707/1; 395/684 :IMAGE AVAILABLE:

3. 4,591,967, May 27, 1986, Distributed drum emulating programmable controller system; Donald A. Mattes, et al., 364/132, 222.2, 222.3, 229, 229.1, 229.4, 230, 230.4, 236.1, 237.2, 237.5, 238.3, 238.4, 245.9, 248, 264, 264.1, 264.2, 265, 266, 270, 270.4, 271, 271.2, 273, 273.1, 273.2, 273.4, 284, 284.3, 284.4, 285, 286, 286.4, DIG.1; 395/290 :IMAGE AVAILABLE:

4. 4,184,202, Jan. 15, 1980, Biorhythm computer; Ian S. McCrae, 364/710.01, 710.02 :IMAGE AVAILABLE:

5. 4,158,285, Jun. 19, 1979, Interactive wristwatch calculator; Edward A. Heinsen, et al., 364/705.07; 341/22; 364/569; 368/29, 109, 111, 251; 968/846, 904, 914, 937, 959, 960, 972, DIG.1 :IMAGE AVAILABLE:

D HIS

```
     (FILE 'USPAT' ENTERED AT 13:35:17 ON 06 NOV 1997)
L1            6 S Y2K OR (2000 PROBLEM)
L2         2141 S (EBCDIC OR ASCII) AND DATE#
L3         1705 S L2 AND FORMAT
L4          790 S L3 AND DIGIT#
L5          719 S L4 AND FIELD#
L6          126 S L5 AND 2000
L7           29 S L6 AND YEAR
```

=> D 1-29

1. 5,668,989, Sep. 16, 1997, Two-**digit** hybrid radix **year** numbers for **year 2000** and beyond; Decao Mao, 707/101; 341/82, 83, 84, 85; 364/744, 771; 395/704, 705; 705/25 :IMAGE AVAILABLE:

2. 5,661,823, Aug. 26, 1997, Image data processing apparatus that automatically sets a data compression rate; Akira Yamauchi, et al., 382/239; 358/430, 433; 382/250, 251 :IMAGE AVAILABLE:

3. 5,644,762, Jul. 1, 1997, Method and apparatus for recording and reading **date** data having coexisting formats; Thomas B. Soeder, 707/6 :IMAGE AVAILABLE:

4. 5,603,081, Feb. 11, 1997, Method for communicating in a wireless communication system; Alex K. Raith, et al., 455/435 :IMAGE AVAILABLE:

5. 5,557,515, Sep. 17, 1996, Computerized system and method for work management; Pamela Abbruzzese, et al., 705/9 :IMAGE AVAILABLE:

6. 5,548,110, Aug. 20, 1996, Optical error-detecting, error-correcting and other coding and processing, particularly for bar codes, and applications therefor such as counterfeit detection; Leonard Storch, et al., 235/472, 462, 494 :IMAGE AVAILABLE:

7. 5,444,820, Aug. 22, 1995, Adaptive system and method for predicting response times in a service environment; Anthony Tzes, et al., 395/22, 11, 23, 61, 900 :IMAGE AVAILABLE:

8. 5,414,838, May 9, 1995, System for extracting historical market information with condition and attributed windows; Anthony D. Kolton, et al., 707/104; 364/282.1, 286.3, DIG.1; 395/117; 705/36 :IMAGE AVAILABLE:

9. 5,371,673, Dec. 6, 1994, Information processing analysis system for sorting and scoring text; David P. Fan, 704/1; 707/531 :IMAGE AVAILABLE:

10. 5,289,362, Feb. 22, 1994, Energy control system; Ronald J. Liebl, et al., 364/140; 165/268; 236/46R; 364/145; 705/412 :IMAGE AVAILABLE:

11. 5,233,513, Aug. 3, 1993, Business modeling, software engineering and prototyping method and apparatus; William P. Doyle, 705/7 :IMAGE AVAILABLE:

12. 5,220,501, Jun. 15, 1993, Method and system for remote delivery of retail banking services; Matthew P. Lawlor, et al., 380/24; 379/93.18; 380/29; 705/43; 902/24 :IMAGE AVAILABLE:

13. 5,113,523, May 12, 1992, High performance computer system; Stephen

R. Colley, et al., 395/800.12; 364/221, 221.7, 231.8, 232.1, 232.8, 235, 237.2, 237.3, 238, 2[illegible]3, 239, 240, 241.9, 242.3, 243 43.4, 243.41, 244, 244.6, 244.9, 24[illegible] 247.1, 247.2, 247.3, 248.1, 2[illegible].2, 252, 259, 268, 268.9, 270, 270.5, 280, 280.1, 280.2, 280.4, 281.3, 281.7, DIG.1 :IMAGE AVAILABLE:

14. 4,977,529, Dec. 11, 1990, Training simulator for a nuclear power plant; Gerald L. Gregg, et al., 364/578, 492; 376/245, 463 :IMAGE AVAILABLE:

15. 4,916,617, Apr. 10, 1990, Controller for well installations; William L. Norwood, 364/422; 137/552.7, 557, 624.2; 166/53, 64 :IMAGE AVAILABLE:

16. 4,768,178, Aug. 30, 1988, High precision radio signal controlled continuously updated digital clock; Charles C. Conklin, et al., 368/47; 375/356; 968/907, 922, DIG.1 :IMAGE AVAILABLE:

17. 4,751,648, Jun. 14, 1988, Local area network data transfer system; Leslie R. Sears, III, et al., 364/422; 340/825.07, 870.01; 364/550, DIG.2; 707/102 :IMAGE AVAILABLE:

18. 4,747,060, May 24, 1988, Data acquisition module and method; Leslie R. Sears, III, et al., 364/481; 340/825.15; 346/33WL; 364/134, 422, 550, 556 :IMAGE AVAILABLE:

19. 4,625,308, Nov. 25, 1986, All digital IDMA dynamic channel allocated satellite communications system and method; Kap S. Kim, et al., 370/321 :IMAGE AVAILABLE:

20. 4,625,276, Nov. 25, 1986, Data logging and transfer system using portable and resident units; William M. Benton, et al., 705/44; 235/379, 380; 379/91.01, 93.18, 106.02, 144, 148, 357; 902/26, 39 :IMAGE AVAILABLE:

21. 4,600,918, Jul. 15, 1986, Equipment for reproduction of alphanumerical data; Pietro Belisomi, et al., 345/160; 340/286.13; 345/152, 168, 507; 348/563; 368/10, 28 :IMAGE AVAILABLE:

22. 4,591,967, May 27, 1986, Distributed drum emulating programmable controller system; Donald A. Mattes, et al., 364/132, 222.2, 222.3, 229, 229.1, 229.4, 230, 230.4, 236.1, 237.2, 237.5, 238.3, 238.4, 245.9, 248, 264, 264.1, 264.2, 265, 266, 270, 270.4, 271, 271.2, 273, 273.1, 273.2, 273.4, 284, 284.3, 284.4, 285, 286, 286.4, DIG.1; 395/290 :IMAGE AVAILABLE:

23. 4,570,217, Feb. 11, 1986, Man machine interface; Bruce S. Allen, et al., 364/188, 191, 921.4, 921.8, 921.9, 926, 926.9, 926.92, 927.3, 927.4, 928, 929.2, 929.3, 935, 935.2, 935.4, 935.41, 940.61, 940.62, 941, 949, 949.3, 959.1, 968, 969, 969.1, 977, DIG.2 :IMAGE AVAILABLE:

24. 4,542,469, Sep. 17, 1985, Programmable demand register with two way communication through an optical port and external reading devices associated therewith; Robert E. Brandyberry, et al., 364/483; 340/870.28; 364/492 :IMAGE AVAILABLE:

25. 4,400,783, Aug. 23, 1983, Event-logging system; Philip F. Locke, Jr., et al., 364/483; 324/113; 364/550, 920, 923, 923.1, 923.2, 926, 926.1, 926.3, 927.8, 927.83, 927.92, 927.94, 927.95, 927.96, 927.99, 929, 929.4, 929.71, 931, 931.1, 932, 932.2, 934, 934.71, 937.1, 937.2, 939, 939.5, 940, 940.1, 940.2, 941, 941.7, 942.7, 942.8, 943.9, 944.7, 946.2, 946.6, 948.4, 948.5, 948.9, 949.3, 950, 950.3, 959.1, 964, 964.1, 965, 965.5, 965.8, 970, 970.5, 975.2, 976, 976.4, DIG.2; 395/551, 557 :IMAGE AVAILABLE:

26. 4,276,597, Jun. 30, 1981, Method and apparatus for information

storage and retrieval; Donald D. Dissly, et al., 707/1; 364/222.2, 222.3, 222.81, 222.9, 225, .4, 225.5, 234, 236.2, 236.3, .4, 236.5, 237.2, 237.3, 248.1, 248.2, .8.3, 252.3, 252.4, 259, 259.2, 9.7, 260.4, 260.6, 260.81, 262.4, 282.1, 282.3, 283.4, 963.1, DIG.1, DIG.2 :IMAGE AVAILABLE:

27. 4,213,174, Jul. 15, 1980, Programmable sequence controller with drum emulation and improved power-down power-up circuitry; Richard E. Morley, et al., 364/145, 184, 919.4, 921, 921.3, 921.8, 926, 926.9, 927.92, 927.99, 932.8, 933.9, 934, 934.1, 935, 935.2, 937, 938, 938.3, 940, 940.1, 940.3, 940.81, 943.9, 944.8, 945.4, 947, 947.5, 948.4, 948.5, 948.9, 949, 950, 950.2, 950.5, 952, 952.2, 969.1, 977.5, DIG.2; 395/750.07 :IMAGE AVAILABLE:

28. 3,887,903, Jun. 3, 1975, Interactive man-machine method and system for grading pattern pieces and for producing an apparel marker; Charles Ronald Martell, 364/470.03, 222, 225, 226.7, 227.3, 234, 234.1, 234.2, 234.4, 235, 236, 237, 237.2, 237.3, 237.4, 237.7, 237.8, 238.3, 238.4, 244, 244.6, 246, 246.3, 248.1, 248.2, DIG.1 :IMAGE AVAILABLE:

29. 3,835,260, Sep. 10, 1974, COMMUNICATION SWITCHING SYSTEM, WITH MARKER, REGISTER, AND OTHER SUBSYSTEMS COORDINATED BY A STORED PROGRAM CENTRAL PROCESSOR; Kenneth E. Prescher, et al., 379/237, 269, 273, 279, 290, 302 :IMAGE AVAILABLE:

D HIS

(FILE 'USPAT' ENTERED AT 13:35:17 ON 06 NOV 1997)
L1 6 S Y2K OR (2000 PROBLEM)

=> D 1-6

1. 5,668,989, Sep. 16, 1997, Two-digit hybrid radix year numbers for year 2000 and beyond; Decao Mao, 707/101; 341/82, 83, 84, 85; 364/744, 771; 395/704, 705; 705/25 :IMAGE AVAILABLE:

2. 5,594,908, Jan. 14, 1997, Computer system having a serial keyboard, a serial display, and a dynamic memory with memory refresh; Gilbert P. Hyatt, 395/887 :IMAGE AVAILABLE:

3. 5,363,096, Nov. 8, 1994, Method and apparatus for encoding-decoding a digital signal; Pierre Duhamel, et al., 341/50, 51 :IMAGE AVAILABLE:

4. 4,993,696, Feb. 19, 1991, Movable stage mechanism; Motomu Furukawa, et al., 269/73, 59 :IMAGE AVAILABLE:

5. 4,984,649, Jan. 15, 1991, Motor vehicle with an automatic limited-slip differential; Heinz Leiber, et al., 180/197, 249 :IMAGE AVAILABLE:

6. 4,862,169, Aug. 29, 1989, Oversampled A/D converter using filtered, cascaded noise shaping modulators; Nicholas R. Van Bavel, et al.,

```
!                  -- Century Conversion --
!
!                  Bruce Dickens   Apr 04, 1996
!
10    open structure tools:name 'otms_src_dir:tools'
      open #2 : name 'last_inv.dat', access output
        print "    Tools "Last Inventory Data Format' Check for 1996 Inventory"
        print "ToolNo                ";  "  Model No  "; "  LAST_INV "; "LAST_INV "
        print "======                ";  "  ========  "; "  ======== "; "======== "
        print "Extract Data:"
        print #2: "ToolNo                ";  "  Model No  "; "  LAST_INV "; "LAST_I
NV "
        print #2: "======                ";  "  ========  "; "  ======== "; "======
== "
        print #2: "Extract Data:"

20    extract structure tools
        yy$ = lpad$ (element$(tools(last_inv),3,"/"), 2, "0" )
        mm$ = lpad$ (element$(tools(last_inv),1,"/"), 2, "0" )
        dd$ = lpad$ (element$(tools(last_inv),2,"/"), 2, "0" )
        cc$= yy$ + "/" +  mm$ + "/" + dd$
        c1$ = change$(cc$,'/','')
        if c1$[1:2] < '50' then
          c$ = '20' + c1$
        else
          c$= '19' + c1$
        end if
!       include c$ < '19960101'
          sort by tools(model)
          sort by rpad$(c$,8, '0')
!      if c$[1:8] < '19960101'  then
        print tools(toolno); tab(23); tools(model);   &
                      tab(35);tools(last_inv); tab(44); c$
        print #2: tools(toolno); tab(23); tools(model);   &
                      tab(35);tools(last_inv); tab(44); c$
                if valid ( c1$, "digits" ) = 0 then
                print; tab(53); " Date format is not digits"
                print #2:; tab(53); " Date format is not digits"
                end if
!               if valid ( c1$, "minlength 6" ) = 0 then
!               print ;tab(50); " Date format is short"
!               print #2: ;tab(50); " Date format is short"
!               end if
                if tools(last_inv) = "" then
                print ;tab(53); " Date format is blank "
                print #2: ;tab(53); " Date format is blank "
                end if
!      end if
30    end extract
        print
        print "Sorted Data:"
        print
40    for each tools
        c1$ = change$(tools(last_inv),'/','')
        print tools(toolno); tab(23); tools(model); &
         tab(35); tools(last_inv); tab(44); c$
        print #2: tools(toolno); tab(23); tools(model); &
         tab(35); tools(last_inv); tab(44); c$
                if valid ( c1$, "digits" ) = 0 then
                print ;tab(53); " Date format is not digits"
                print #2: ;tab(53); " Date format is not digits"
                end if
!               if valid ( c1$, "minlength 6" ) = 0 then
!               print ;tab(53); " Date format is short"
!               print #2: ;tab(53); " Date format is short"
!               end if
```

Exhibit A

T NUMBER

APPLICATION SERIAL NUMBER

08/723,574

APPLICANT'S NAME (PLEASE PRINT)

BRUCE DICKENS

IF REISSUE, ORIGINAL PATENT NUMBER

INTERNATIONAL CLASSIFICATION

G06F 17/30

ORIGINAL CLASSIFICATION

CLASS	SUBCLASS
707	6

CROSS REFERENCE(S)

CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)		
707	102	7	202

GROUP ART UNIT 2771

ASSISTANT EXAMINER (PLEASE STAMP OR PRINT FULL NAME)

PRIMARY EXAMINER (PLEASE STAMP OR PRINT FULL NAME)

WAYNE AMSBURY

ISSUE CLASSIFICATION SLIP

270 (5-91)

U.S. DEPARTMENT OF
PATENT AND TRADE

08725574

PATENT APPLICATION

08725574

APPROVED FOR LICENSE

INITIALS

NOV 0 6 1996

Date Entered or Counted	CONTENTS	Date Received or Mailed
	1. Application 1 papers.	
	2. Revocation / PA	1/31/97
	3. I.D.S.	4-16-97
	4. Notice of Acceptance	10/22/97
11/7	5. Rej (3 mos)	11/17/97
	6. Ext of time 1 month	3-20-98
	7. Recon.	3-20-98
	8. Supp IDS	3-20-98
	9. supp amdt A	4-2-98
4-3	10. Ex's Amdt B	4-8-98
6/23/98	11. Formal Drawings (1 shts) set 1	5/14/98
	12. PTO GRANT SEP 0 8 1998	
	13. Req for CofC	10-14-98
	14.	
	15.	
	16.	
	17.	
	18.	
	19.	
	20.	
	21.	
	22.	
	23.	
	24.	

File Number: S370-40
Program Number: 5688-198

Printed in U.S.A.

IBM SAA AD/Cycle Language Environment/370 Publications

GC26-4785	Fact Sheet
GC26-4786	Concepts Guide
SC26-4817	Planning for Installation and Customization
SC26-4818	Programming Guide
SC26-4829	Debugging and Run-Time Messages Guide
LY37-3711	Diagnosis Guide
GC26-4774	Licensed Program Specifications

SC26-4818-00

Readers' Comments
SC26-4818-00

IBM®

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department J58
PO Box 49023
San Jose, 95161-9945

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Readers' Comments

IBM SAA AD/Cycle Language Environment/370
Programming Guide
Version 1 Release 1

Publication No. SC26-4818-00

Please use this form to tell us what you think about the accuracy, clarity, organization, and appearance of this manual. Any suggestions you have for its improvement will be welcome.

In sending information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

If you would like a reply, be sure to print your name, and your address or phone number below.

If you prefer to send comments by Fax, use this U.S. number: (408) 463-3114

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address listed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Name	Address
Company or Organization	
Phone No.	

TXTLIB (CMS) 54

U

user comments 459
user condition handler 85, 86, 89, 91, 96
user exit 9, 23—24, 173—184, 450, 465
  assembler 23, 127, 128, 173, 174, 207
  for initialization 173, 189
  for termination 173, 174, 189
  global 174
  HLL 23, 173, 183
  under CICS
user-server environment 450

run-time options *(continued)*
COUNTRY 115, 214, 239, 241, 360
syntax description 224
DEBUG 214
syntax description 227
ENV 214
syntax description 228
ERRCOUNT 90, 214, 312
syntax description 229
EXECOPS 17—19, 43, 46, 48, 56, 57, 123, 194, 208, 214, 229
FLOW 214
syntax description 230
HEAP 76, 78, 123, 214, 218, 247, 268, 274, 277, 279
syntax description 230
INTERRUPT 123, 214
syntax description 232
LIBSTACK 123, 215
syntax description 233
MSGFILE 94, 110—111, 123, 124, 129, 154, 203, 215, 217, 224
syntax description 234
MSGQ 235
syntax description 235
NATLANG 115, 215, 360
syntax description 235
PLIST 17—19, 215
syntax description 236
REDIR 215
syntax description 238
RPTOPT 215
syntax description 239
RPTSTG 75, 215, 220, 222, 232, 246, 280, 281
syntax description 240
RTEREUS 123, 185
syntax description 243
SIMVRD 123, 215
syntax description 244
STACK 74, 75, 123, 215, 218
syntax description 245
STORAGE 123, 215, 271, 277, 280
syntax description 246
TERMTHDACT 215
syntax description 249
TEST 84, 216, 233
syntax description 250
TRAP 87, 88, 128, 175, 216, 217
syntax description 252
UPSI 216
syntax description 253
VCTRSAVE 216
syntax description 253
XUFLOW 98, 216
syntax description 254
run-unit 5, 143
run-unit (CICS) 120, 124
running under CMS
specifying run-time options 206
running under MVS 41—43
specifying run-time options 42, 206
writing JCL to run an application 41, 42
running under TSO 48—49
specifying run-time options 48, 206

S

S-names 430—433, 440—443, 452
save area 83
SCEELKED link library 31, 33, 38, 39, 44, 46, 64, 65, 66, 446
SCEERUN load library 33, 38, 64, 66
scope of variables 6, 148, 149, 152, 153
setlocale() 225
Showa era 367
signal() 90—94
See also condition handling
SORT/MERGE 464—466
SQL 135
stack frame 83—89, 90, 92—96, 422, 424
obtaining the 83
stack frame zero 84, 87, 94, 96
stack storage 8, 73—75
initial stack segment 74, 75, 246
stack increment 74, 75
tuning the 75
standard streams 114, 462
START command (CMS) 56, 206
stderr 113—114, 462
stdout 113—114
STOP RUN statement 20, 151, 172, 174, 189, 193
STOP statement 189, 195
storage 246, 267—286
management model 73—78
operating services for, 170
report 220, 222, 232, 240, 246, 281
service routines for, 198
tuning 219, 221, 230, 233, 240, 245, 246, 280
subroutine 6, 143, 159, 165, 185, 202
symbol information 459
System Programming Facility 444—451

T

Taisho era 367
termination 90, 138, 150, 159, 161, 173, 174, 185, 313
Termination Imminent 229, 312
thread 7
multiple 7
timestamp 371, 391

nested condition 97, 235, 315, 316
nested enclave 111, 201—203, 465
See also CICS, EXEC CICS LINK command
See also CICS, EXEC CICS RETURN command
See also CICS, EXEC CICS XCTL command
NSS (Named Shared Segment) 27, 28, 51, 186, 190, 191

O

Object Library Utility 452—459
library utility map 456
ommitted parameter 82, 261, 428
ON EXCEPTION clause 90
ON SIZE ERROR clause 90, 95
OS ATTACH command 172
OS LINK command 172
OSRUN command (CMS) 57, 206, 237
out-of-storage condition 248
overflow condition 91, 95, 98, 319

P

parameter
See also ommitted parameter
list 146, 159, 415
list format 10—19, 178, 194, 236
passing 9, 428
PARM statement 32, 35, 38
percolate, action 85, 87, 89, 92, 94, 96, 99
picture string 360, 363—371, 391—404
defaults 468—474
pragmas 122
#pragma 61, 141—142
#pragma comment 459
#pragma csect 432, 452
#pragma map 432
#pragma runopts 14, 42, 132, 207, 208, 461
pre-initialization facility 185—201, 244
CEEPIPI(add_entry) 186, 196
CEEPIPI(call_main) 186, 188, 192
CEEPIPI(call_sub) 186, 194
CEEPIPI(init_main) 186, 190
CEEPIPI(init_sub) 186, 191
CEEPIPI(term) 186, 195
CEEXPIT macro 187
CEEXPITS macro 187
CEEXPITY macro 187
service rouitnes for, 198
prelink facility 29, 31, 148, 430—443, 449
input 430
invoking under MVS 434
invoking under TSO 435
invoking under VM/CMS 434
prelink options 436
utility map 437
process 5
program control table (CICS) 121
program interrupts 83, 87, 98, 175, 252, 254, 312, 320, 465
nested enclave 203
under CICS 126, 128
under IMS 133
program model 4
program processing table (CICS) 120, 121
Program Prolog Area (PPA) 160, 162
promote action 86, 87, 89, 99

R

raise() 91, 92, 151, 184
random numbers 425
reason code 20—22
in user exits 177
under CICS 127
under IMS 134
recursion 99, 202
reentrancy 27—29, 134, 152, 175, 188, 446, 448
region (CICS) 120
register save area 83, 247
release() 152, 153
RENAME control statement 433, 440, 442
RENT compile-time option 148, 430, 449
Republic of China era 360, 367
resume action 85, 89, 94, 99
resume cursor 85, 87, 88, 94, 97, 300
return code 20—22
in user exits 177
RETURN-CODE special register 21
under IMS 134
return/reason code 9, 20—22, 175, 193, 194
run-time options 2, 173, 179, 191, 192, 202, 206—212, 213—254
ABPERC 88, 123, 213
syntax description 216
AIXBLD 123, 213
syntax description 217
ALL31 213, 231, 246
syntax description 218
ANYHEAP 123, 213
syntax description 219
ARGPARSE 208
syntax description 220
BELOWHEAP 123, 214
syntax description 221
CBLOPTS 43, 194, 208, 214, 467
syntax description 222
CBLPSHPOP 128, 214
syntax description 223
CBLQDA 123, 214
syntax description 223
CHECK 214
syntax description 224

instance specific information (ISI) 81, 235, 288, 291, 308, 311
interlanguage communication (ILC) 2, 9, 138—139, 140—156, 160
  special declarations 141
intrinsic functions 77, 383, 412
ISPF 464, 467

J

Japanese eras 360, 367

L

L-names 430—433, 437, 439, 440—443, 452, 454
language-specific condition handler 253
LIBRARY statement 34, 37, 440, 441
lilian date 359, 361, 368, 374, 376, 382, 390
LINK command (TSO) 44, 48
link pack area 28, 40, 47, 190, 191
link-editing under CMS 50—55
  options 51—53
link-editing under MVS 30—36
  detecting errors 36
  including additional modules as input 34
  input to the linkage editor 31
  old objects 3
  options 32, 35
  standard data sets for, 32—33
  writing JCL for the linkage editor 34
link-editing under TSO
  including additional modules as input 53
  input to the linkage editor 44
LKED command (CMS) 50, 56, 57
  syntax description 54
LOAD command (CMS) 50, 53, 56, 154, 441, 444, 445, 450
  syntax description 51
LOADGO command (TSO) 45, 47, 48
loading under MVS 37—40
  input to the loader 37
  options 40
  standard data sets for, 39
  writing JCL for the loader 37, 38, 39
loading under TSO 44—47
  options 46—47
LOADLIB (CMS) 51, 54, 56, 57, 58, 154
local data 6, 28
long name support 442, 452
LONGNAME compile-time option 430, 454

M

main routine 5, 185, 190, 202
  assembler main 164
  determining the 6, 143
main routine *(continued)*
  return from 20
mapping of l-names to s-names 430, 432
math library 405—414
  CEExABS 408
  CEExACS 408
  CEExASN 408
  CEExATH 408
  CEExATN 408
  CEExAT2 408
  CEExCOS 408
  CEExCSH 408
  CEExCTN 408
  CEExDIM 409
  CEExERF 409
  CEExEXP 408
  CEExGMA 409
  CEExINT 409
  CEExLGM 409
  CEExLG1 408
  CEExLG2 408
  CEExLOG 408
  CEExMOD 409
  CEExNIN 409
  CEExNWN 409
  CEExSGN 409
  CEExSIN 408
  CEExSNH 408
  CEExSQT 409
  CEExTAN 408
  CEExTNH 408
  CEExXPD 408
  CEExXPI 408
  CEExXPS 408
Meiji era 367
message file 6, 8, 80, 94, 110—118, 123, 154, 203, 234
message handling 79, 80, 110—118, 234, 235
  See also instance specific information (ISI)
  callable services for, 327—337
MinKow era 367
multitasking 154, 460—463
  linking 460, 461
  main task program load module 460
  parallel functions load module 460
  running 462
  specifying run-time options 461
MVS 9
MVS parameter list format 12

N

National Language Support 110, 115, 235, 360
  callable services for 338—358
  defaults 468—474

compatibility *(continued)*
C/370 185, 208
ILC 131, 140
run-time option 255—257
compilation unit 5, 84
condition handler 8
condition handling 81, 83—109, 248, 405
See also ?
callable services for, 291, 294, 296, 298, 300, 306, 311, 316, 318
COBOL/370 semantics 90, 94—95
condition manager 84
C/370 semantics 90—94
default actions 89, 94
for ILC applications 95—96, 153
handled condition 96
HLL-specific condition handler 85, 88, 92, 253, 312, 465
in assembler routines 159
in nested enclaves 202
in user exits 128
severity 90, 94, 229, 249, 250, 312
under CICS 127, 223
unhandled condition 21, 89
user-written condition handler 85, 88, 92, 97, 98, 295, 297, 315, 320, 465
condition token 79—82, 110, 287, 298, 306
CPLINK CLIST 435
CPLINK EXEC 434, 446
C370LIB CLIST 455
C370LIB EXEC 452

D

data type definitions 144—148, 262, 428
database rollback 128, 134
DATA24 option (COBOL) 133
date and time services 359—361, 368, 371, 374, 376, 378, 380, 385, 387, 389, 391
DB/2 135
debugging 83, 84, 94, 227, 250, 418
DFHECI 122
DFHELII 121, 125
DFSORT facility 464
DISPLAY statement 112, 116
DSA (dynamic storage area) 75, 84, 159, 248
dumps 171, 178, 224, 249, 314
CEE3DMP callable service 420
under Assembler 174
under CICS 123, 124, 129

E

enclave 5
multiple 7
EXEC statement (MVS) 32, 35, 38, 40, 41
exit() function 6, 93, 151, 174, 184, 189, 446
external data 6, 28, 29, 96, 148—150, 152, 153, 188
name scope of 148
name space of 150

F

feedback code 428
FEEDBACK data type 263
in callable services 81, 261
in math routines 410
omitting 82, 261
fetch() 96, 151—153
file map 439
FILEDEF command (CMS) 54, 112, 173
freestanding application 444—451
function call (C/370) 84, 96

G

GENMOD command (CMS) 50, 53, 56, 57, 444, 445
GLOBAL command (CMS) 50, 54, 56, 57, 444
global error table 90
See also condition handling
GOBACK statement 20, 151, 244

H

handle cursor 87, 88, 99, 300
heap storage 6, 8, 75, 78, 230
callable services for 267—286
heap element 76, 270, 271
heap increment 76, 231
initial heap segment 76, 231, 279
under CICS 123
under IMS 133
Heisei era 367

I

IGZERRE 185
ILBOSTP0 185
IMS 9, 28, 131—134, 228
CEETDLI interface 131
condition handling under 133
parameter list format 13
storage considerations 133
INCLUDE command (CMS) 50, 53, 175, 431
INCLUDE statement (MVS) 32, 34, 37, 175, 431, 440, 445, 447
initialization 2, 158, 173, 179, 185, 444
alternate initialization routines 444, 445
initialization routines 27, 33
nested enclave 173, 176, 180—181, 202
input/output (I/O) 121, 124, 153, 221, 464

callable services *(continued)*
  CEEQCEN 361
    syntax description 385
  CEERAN0
    syntax description 425
  CEESCEN 361
    syntax description 387
  CEESECI 382
    syntax description 389
  CEESECS 372, 380
    syntax description 391
  CEESGL 82, 83, 88, 90, 92, 171, 252, 280
    syntax description 311
  CEETEST 84, 251
    syntax description 418
  CEEUTC
    syntax description 395
  CEE3ABD 171
    syntax description 313
  CEE3CNC
    syntax description 315
  CEE3CTY 115, 225
    syntax description 345
  CEE3DMP 124, 129, 171
    syntax description 420
  CEE3GRN
    syntax description 316
  CEE3LNG 115, 236
    syntax description 348
  CEE3MCS
    syntax description 352
  CEE3MDS
    syntax description 339
  CEE3MTS
    syntax description 354
  CEE3PRM 159
    syntax description 415
  CEE3RPH
    syntax description 281
  CEE3SPM 98, 428
    syntax description 318
  CEE3USR
    syntax description 416
  guidelines for writing 428—429
  invoking 170, 261—266
  invoking under COBOL 261, 333
  invoking under C/370 261
calls to other routines
  dynamic call 27, 56, 124, 125, 140, 151, 153, 154
  static call 27, 124, 125, 140, 142
cancel() 153
catalogued procedures (MVS) 31, 38, 42, 62—71
  CEEWG 63, 65
  CEEWL 65
  CEEWLG 63
  data set names 64

catalogued procedures (MVS) *(continued)*
  ECDLIB 454
  EDCCL 63, 460
  EDCCLG 63
  EDCPL 63, 434
  EDCPLG 63, 434
  IGYWCG 63
  IGYWCL 63
  IGYWCLG 63
  invoking 62
  modifying 70
  overriding and adding DD statements 70
  overriding and adding EXEC statements 70
  overriding default options 62
  step names 62, 63
  unit names 64
CEEBINT user exit 24, 173, 183
CEEBXITA user exit 24, 173, 174
CEECOPT csect 122, 206
CEEDOPT csect 17, 42, 206, 209, 225, 236, 350
CEEUOPT csect 17, 42, 206, 209, 225, 236, 350
CEEXOPT macro 206, 209—212
century window 360, 385, 387
CESE transient data queue 111, 124, 129, 130, 234, 250, 424
ChuHwaMinKow era 367
CICS 9, 28, 120—130
  callable service behavior under, 124, 279, 314
  condition handling under, 126—127
  EXEC CICS LINK command 120, 124, 202
  EXEC CICS RETURN command
  EXEC CICS XCTL command, 120, 124, 202
  ILC 124—125
  message handling under, 111, 129
  parameter list format 14
  run-time option behavior under, 122—124, 217, 220, 222, 224, 230, 232, 233, 234, 244, 246, 248, 252
  storage management under, 126
  terminology 120
  transaction 120, 121
CLISTs (TSO) 435, 455
CMOD EXEC 434
CMS 9
  dynamic calls 154, 186
  dynamically loaded routines 27
  link-editing under 50—55
  parameter list format 12, 58
  pre-linking under 434
  running under 55—61
  using system programming facilities 445
common anchor area (CAA) 159, 246
common environment 2
compatibility 3, 142, 153, 208
  assembler 158
  COBOL 124, 125, 202, 208, 223

Index

A

ABENDs 83, 85, 87, 97, 127, 171, 195, 203, 229, 252, 276, 280, 313
  ABEND codes 126, 127, 177, 312, 423
  in nested enclaves 202
  percoloating 97, 175, 179, 216
  requesting 174, 178
  specifying in user exits 97, 128, 175
  system ABENDs 97, 179, 216
  under CICS 126, 127, 128, 202
  under IMS 133
  user ABENDs 97, 179, 216
  when using SORT 466
abort() function 22, 91, 93, 151, 184, 189
ACCEPT statement 112, 117
AD/Cycle
  definition
  LE/370 position in
AD/Cycle CODE/370 174, 418
ALLOCATE command (TSO) 47, 49
AMODE considerations 77, 159, 218, 232, 262
  C/370 124, 154
  for CEEBXITA user exit 175
  ILC 138
  in pre-initialized routines 185
  under CICS 279
argument 10
  list pointer 11
ASSEMBLER macros
  CEECAA 161, 162
  CEEDSA 161, 162
  CEEENTRY 160
  CEEPPA 161, 162
  CEETERM 161
ASSEMBLER routines 125, 158—172
  callable services for, 170
  calling conventions for, 158
  condition handling for, 159
  operating services for, 170, 428
  termination processing 159, 161
atexit list 151, 189
ATTACH commands 172
automatic call library 32, 34, 35, 39, 431
automatic data 6

C

CALL command (TSO) 48, 49
CALL statement (COBOL) 6, 82, 151
callable services 3
  CEECRHP 274, 280
    syntax description 267

callable services *(continued)*
  CEECZST
    syntax description 270
  CEEDATE 363
    syntax description 368
  CEEDATM 389, 393
    syntax description 371
  CEEDAYS 369, 387
    syntax description 361
  CEEDCOD 80
    syntax description 287
  CEEDSHP 267, 280
    syntax description 274
  CEEDYWK
    syntax description 374
  CEEFMDA
    syntax description 338
  CEEFMTM
    syntax description 343
  CEEFRST 280
    syntax description 276
  CEEGMT
    syntax description 376
  CEEGMTO
    syntax description 378
  CEEGPID
    syntax description 290
  CEEGQDT
    syntax description 291
  CEEGTST 271
    syntax description 278
  CEEHDLR 85, 86, 89, 92, 94, 99, 159, 171, 252, 297, 315
    syntax description 294
  CEEHDLU 86, 99, 171
    syntax description 296
  CEEISEC 380, 390
    syntax description 380
  CEEITOK
    syntax description 298
  CEELOCT
    syntax description 383
  CEEMGET 82
    syntax description 327
  CEEMOUT 82, 124, 171
    syntax description 330
  CEEMRCR 85, 87
    syntax description 300
  CEEMSG 82, 235
    syntax description 331
  CEENCOD 80, 287
    syntax description 306

© Copyright IBM Corp. 1991

W

word. A contiguous series of 32 bits (4 bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four.

working storage. In COBOL/370, the storage required for data items in the Working_Storage section.

Z

zoned decimal format. Synonym for *unpacked decimal format.*

source program. A set of instructions written in a programming language that must be translated to machine language before the program can be run.

stack. An area of storage used for suballocation of stack frames (DSAs). Such suballocations are allocated and freed on a LIFO (last in, first out) basis. A stack is a collection of one or more stack segments consisting of the initial stack segment and zero or more increments.

stack frame. The physical representation of the activation of a routine. The stack frame is allocated on a LIFO stack and contains various pieces of information including a save area, fields to assist the acquisition of a stack frame from the stack, and the local, automatic variables for the routine. Synonymous with *DSA*.

stack frame zero. The conceptual stack frame just prior to the first stack frame on the stack.

stack segment. A relatively large, contiguous area of storage obtained directly from the operating system. The LE/370 storage management scheme subdivides stack segments into individual DSAs. If the initial stack segment becomes full, a second segment or increment is obtained from the operating system.

stack storage. See *stack* and *automatic storage*.

standard system action. The name given to the language-defined default action taken when a condition occurs and it is not handled by a condition handler.

statement. In programming languages, a language construct that represents a step in a sequence of actions or a set of declarations.

static call. A call that results in the resolution of the called program statically at link-edit time. Contrast with *dynamic call*.

static data. Data that retains its last-used state across calls.

static storage. Storage that persists and retains its value across calls. Contrast with *dynamic storage*.

subroutine. In general, any routine within an application called by another routine.

subsystem. A secondary or subordinate system, or programming support, usually capable of operating independently of or asynchronously with a controlling system. Examples: CICS, IMS.

syntax. The rules governing the structure of a programming language and the construction of a statement in a programming language.

T

thread. The basic line of execution within the LE/370 program management model. It is dispatched by the system with its own instruction counter and registers. Threads may execute concurrently with other threads. The thread is where actual code resides. Note: In the first release of LE/370, one thread per enclave is supported.

token. See *condition token*.

transient data queue. A file to which run-time messages are written under CICS. Under LE/370, the name of this file is CESE.

TSO. Time sharing option.

U

unpacked decimal format. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte. Bits 0 through 3 of all other bytes contain 1s (hex F). For example, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Synonymous with *zoned decimal format*.

user condition handler. A routine established via the CEEHDLR callable service to handle a condition or conditions when they occur in the common run-time environment. A queue of user condition handlers established via CEEHDLR may be associated with each stack frame in which they are established.

user exit. A routine that takes control at a specific point in an application. Two assembler user exists and one HLL user exit are provided by LE/370. They are invoked to perform initialization functions and both normal and abnormal termination functions.

user heap. Synonymous with *initial heap segment*

user stack. An independent area of stack storage that may be located above or below 16M, designed to be used by both LE/370 library routines and compiled code. See also *stack, stack frame, library stack*.

V

vendor. A person or company that provides a service or product to another person or company.

promote. To change a condition. A condition is promoted when a condition handling routine changes the condition to a different one. A condition handling routine promotes a condition because the error needs to be handled in a way other than that suggested by the original condition.

PSW. Program status word.

Q

qdata. Qualifying data. A list of addresses that a user-written condition handler established for a math routine can use to identify and react to a given instance of a condition.

R

reason code. A value returned to the invoker of an enclave which indicates how the enclave terminated. The value reflects whether the enclave terminated successfully, or unsuccessfully due to an unhandled condition.

recursive routine. A routine that can call itself or be called by another routine which it has called.

reentrant. The attribute of a routine or application that allows more than one user to share a single copy of a load module.

register. (1) Special processing areas that hold a specific amount of data and can process, load, and store this data quickly. (2) To specify formally. In LE/370, to register a condition handler means to add a user-written condition handler onto a routine's stack frame using the CEEHDLR callable service.

relocatable load module. Under CMS, a combination of object modules having cross references resolved and prepared for loading into storage for execution.

resident routines. A category of LE/370 library routines linked with your application. They include such things as initialization routines and *callable service stubs*.

resume. To begin execution in an application at the point immediately after which a condition occurred. A resume occurs when the condition manager determines that a condition has been handled and normal application execution should continue.

resume cursor. Designates the point in the application where a condition occurred when it is first reported to the condition manager. The resume cursor also designates the point where execution resumes after a condition is handled, usually at the instruction in the application immediately following the point at which the error occurred. The resume cursor can be moved via the CEEMRCR callable service.

return code. A code produced by a routine to indicate its success. It may be used to influence the execution of succeeding instructions.

return_code_modifier. A value set by LE/370 routines that manage the enviroment. It indicates whether or not an enclave terminated successfully.

root load module. The load module containing a main routine and the first to be executed in an application.

routine. In this book, used as an exact equivalent of a COBOL/370 *compilation unit* or C/370 function or program, and means a named external routine, with or without named entry points, and with or without internal (contained) routines.

run. To cause a program, utility, or other machine function to be performed.

run time. Any instant at which a program is being executed. Synonymous with *execution time*.

run-time environment. A set of resources that are used to support the execution of a program. Synonymous with *execution environment*.

run unit. One or more object programs that are executed together. In LE/370, a run unit is the equivalent of an *enclave*.

S

scalar. A quantity characterized by a single value. Contrast with *aggregate*.

scope. The portion of an application within which the definition of a variable remains unchanged.

segment. See *stack segment*.

signal. In C/370, signals are conditions that may or may not be reported during program execution, depending upon how they are defined to the condition handler. A condition is registered in C/370 using the signal() function; a condition is raised using the raise() function.

single-precision. Pertaining to the use of one computer word to represent a number in accordance with the required precision. Needed for proper alignment. See also *precision, double-precision*.

source code. The input to a compiler or assembler, written in a source language.

O

object code. Output from a compiler or assembler which is itself executable machine code or is suitable for processing to produce executable machine code.

object module. A portion of an object program suitable as input to a linkage editor. Synonymous with *text deck, object deck.*

operating system. Software that controls the running of applications; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

output procedure. A set of statements, to which control is given during the execution of a SORT statement after the sort function is completed, or during the MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

overflow. That portion of an operation that exceeds the capacity of the intended unit of storage.

owning stack frame. Given the calling sequence of Routine 1 calling Routine 2 that in turn calls Routine 3, Routine 3 is the owning stack frame if a condition occurs while Procedure 3 is executing.

P

packed decimal format. A format in which each byte in a field except the rightmost digit represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111.

pad. To fill unused positions in a field with dummy data, usually zeros, ones, or blanks.

parameter. Data items that are received by a routine.

pass by content. A COBOL argument passing style synomymous with passing an argument by value directly. In this style, R1 contains a pointer to a copy of the argument.

pass by reference. In programming languages, one of the basic argument passing semantics. The address of the object is passed. Any changes made by the callee to the argument value will be reflected in the calling routine at the time the change is made.

pass by value. In programming languages, one of the basic argument passing semantics. The value of the object is passed. Any changes made by the callee to the argument value will not be reflected in the calling routine.

percolate. A response that occurs when the condition handler declines to handle a condition, allowing it to be handled by a lower level handler.

picture string. Character strings used to specify date and time formats.

pointer. A data element that indicates the location of another data element.

portability. The ability to transfer an application from one platform to another with relatively few changes to the source code.

precedence. In programming languages, an order relation defining the sequence of the application of operations or options.

precision. A measure of the ability to distinguish between nearly equal values. See *single-precision* and *double-precision.*

preinitialization. A facility that allows a routine to initialize the run-time environment once, perform multiple executions within the enviroment, then explicitly terminate the environment.

prelinker. A utility that concatenates compile-time initialization information information from one or more object modules into a single initialization unit. In the process, the static external data part is mapped.

preprocessor. A routine that examines application source code for preprocessor statements that are then executed, resulting in the alteration of the source.

procedure. A named block of code that can be invoked, usually via a CALL. In LE/370, the term *routine* is used as generic for a procedure or a function.

process. The highest level of the LE/370 program management model. A process is a collection of resources, both program code and data, and consists of at least one enclave.

program mask. A structure that describes the manner in which S/370 hardware detected conditions are to be handled.

program status word (PSW). An area in storage used to indicate the order in which instructions are executed and to hold and indicate the status of the operating system. The program mask (bits 20 and 23) of the PSW can be manipulated to enable or disable the detection of some hardware conditions under LE/370

prolog. The code sequence when a routine is entered.

J

job control language (JCL). A sequence of commands used to identify a job to an operating system and to describe a job's requirements.

L

Language Environment/370. An IBM software product that provides a common run-time environment and common run-time services for all IBM SAA High Level Language compilers.

LE/370-conforming. Adhering to LE/370's common interface.

language-specific condition handler. A set of HLL-specific routines which enforce language condition handling semantics for a given stack frame. A language-specific handler is automatically associated with a stack frame when the stack frame is allocated.

library. A collection of functions, subroutines, or other data.

library stack. An independent area of stack storage, allocated below the 16M line, designed to be used only by LE/370 library routines. See also *stack, user stack, stack frame.*

link pack area (LPA). In MVS, an area of main storage containing reenterable routines from system libraries. Their presence in main storage saves loading time when one is needed.

linkage editor. A program that resolves cross-references between separately assembled object modules and then assigns final addresses to create a single relocatable load module. The linkage editor then stores the load module in a program library in main storage.

link-edit. To create a loadable computer program by means of a linkage editor.

load module. An application or routine in a form suitable for execution. The routine has been compiled and link-edited; that is, address constants have been resolved.

local data. Data that is known only to the routine in which it is declared. Equivalent to local data in C and working storage in COBOL.

M

main routine. The first routine in an enclave to gain control from the invoker.

megabyte (M). 1,048,576 bytes.

multitasking. See *multithreading.*

multithreading. Mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks, or threads.

MVS. Multiple Virtual Storage operating system.

N

name scope. The portion of an application within which a particular declaration of external data applies or is known.

name space. The portion of a load module within which a particular declaration of external data applies or is known.

National Language Support (NLS). Translation requirements affecting parts of licensed programs; for example, translation of message text and conversion of symbols specific to countries.

natural reentrancy. The attribute of applications that contain no external data and do not require additional processing to make them reentrant. Contrast with *constructed reentrancy.*

nested condition. A condition that occurs during the handling of another, previous condition. Nested conditions are tolerated by LE/370 only in some circumstances. For example, you can use the CEE3CNC callable service to allow nested conditions within a user-registered condition handler.

nested enclave. A new enclave created by an existing exclave. The nested enclave that is created must be a new main routine within the process.

next sequential instruction. The next instruction of a routine that will be executed in the absence of any branch or transfer of control.

NLS. National Language Support.

function. A routine that is invoked by coding its name in an expression. The routine passes a result back to the invoker through the routine name.

G

G.E.T.. Global error table

global error table (G.E.T.). A method employed by some HLLs, for example, C/370, to determine actions for handling conditions. Whereas LE/370 condition handling actions are defined at the stack frame level, actions defined via the global error table apply to an entire application unless explicitly changed.

H

handle cursor. A cursor used by the condition manager as it traverses the stack. The handle cursor points to the condition handler currently being invoked for the current the stack frame on the stack.

heap 0. Synonymous with *initial heap*

heap. An area of storage used for allocation of storage whose lifetimes are not related to the execution of the current routine. The heap consists of the initial heap segment and zero or more increments. See *anywhere heap, below heap, initial heap,* and *additional heap.*

heap element. A contiguous area of storage allocated by a call to the CEEGTST service. Heap elements are always allocated within a single heap segment.

heap increment. See *increment.*

heap segment. A relatively large, contiguous area of storage obtained directly from the operating system. The LE/370 storage management scheme subdivides heap segments into individual heap elements. If the initial heap segment becomes full, LE/370 obtains a second segment, or increment, from the operating system.

heap storage. See *heap.*

High Level Language (HLL). A programming language above the level of assembler language and below that of program generators and query languages. In the first release of LE/370, the supported HLLs are C/370 and COBOL/370.

I

ILC. Interlanguage communication.

IMS. Information Management System, IBM licensed product. IMS supports hierarchical databases, data communication, translation processing, database backout and recovery.

increment. The second and subsequent segments of storage allocated to the stack or heap.

indirect parameter passing. Placing an address in a parameter list. In other words, passing a pointer to a value instead of passing the value itself.

initial heap. The LE/370 heap controlled by the HEAP run-time option and designated by a *heap_id* of 0. The initial heap contains dynamically allocated user data. See also *additional heap, anywhere heap, below heap.*

initial heap segment. The first heap segment. A heap consists of the initial heap segment and zero or more additional segments or increments.

initial stack segment. The first stack segment. A stack consists of the initial stack segment and zero or more additional segments or increments.

input procedure. A set of statements, to which control is given during the execution of a SORT statement, for the purpose of controlling the release of specified records to be sorted.

instance specific information (ISI). Located within the LE/370 condition token, the ISI contains information used by the condition manager to identify and react to a specific occurrence of a condition.

integer. A positive or negative whole number or zero.

interactive. Pertaining to a routine or system that alternately accepts input and responds. In an interactive system, a constant dialog exists between user and system. Contrast with *batch.*

interlanguage communication (ILC). The ability of routines written in different programming languages to communicate. ILC support allows the application writer to readily build applications from component routines written in a variety of languages.

interrupt. A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.

ISI. Instance Specific Information.

ddname. Data definition name. The logical name of a file within an application. The ddname provides the means for the logical file to be connected to the physical file.

default. A value that is used when no alternative is specified.

DFSORT. A product used to sort and merge records in a specified sequence.

direct parameter passing. Placing a value directly in the parameter list body.

disabled/enabled. See *enabled/disabled*.

double byte character set (DBCS). A collection of characters represented by a 2-byte code.

double-precision. Pertaining to the use of two computer words to represent a number in accordance with the required precision. See also *precision, single-precision*.

DSA. Dynamic storage area. Synonym for *stack frame*.

dynamic call. A call that results in the resolution of the called routine at run time. Contrast with *static call*.

dynamic storage. Storage acquired as needed at run time. Contrast with *static storage*.

dynamic storage area (DSA). An area of storage obtained during the running of an application which consists of a register save area and an area for automatic data, such as program variables. DSAs are generally allocated within LE/370-managed stack segments. DSAs are added to the stack when a routine is entered and removed upon exit in a LIFO (last-in, first-out) manner. Synonym for *stack frame*.

E

EBCDIC. Extended binary-coded decimal interchange code.

EIB (EXEC interface block). In CICS, a control block containing information useful in the execution of an application, such as a transaction identifier and a time and a date when the transaction is started.

enabled/disabled. A condition is enabled when its occurrence will result in the execution of condition handlers or in the performance of a standard system action to handle the condition as defined by LE/370.

A condition is disabled when its occurrence will apparently be ignored by the condition manager.

enclave. In LE/370, an independent collection of routines, one of which is designated as the main routine. An enclave is roughly analogous to a program or rununit.

entry name. In Assembler language, a programmer-specified name within a control section that identifies an entry point, and can be referred to by any control section.

entry point. In Assembler language, the address or label of the first instruction that is executed when a routine is entered for execution.

environment. A set of services and data available to a program during execution. In LE/370, environment is normally a reference to the run-time environment of HLLs at the enclave level.

epilog. Code generated at the end of a routine, normally causing a return to the caller of the routine.

external data. Data that persists over the lifetime of an enclave and maintains last-used values whenever a routine within the enclave is reentered.

execution time. Synonym for *run time*.

execution environment. Synonym for *run-time environment*.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 8-bit characters.

external data. Data that is known throughout the enclave and represents a piece of storage, independent of the number of source files comprising the enclave. Within an enclave consisting of a single load module, it is equivalent to C static external data and COBOL external data.

external routine. A procedure or function that may be invoked from outside the program in which the routine is defined.

F

feedback code (fc). A condition token value. If you specify *fc* in a call to a callable service, a condition token indicating whether the service completed successfully is returned to the calling routine.

file. A named collection of related data records that is stored and retrieved by an assigned name. Synonymous with *data set*.

file definition statement (FILEDEF). In CMS, serves as the connection between the logical name of a file and the physical name.

C

callable services. A set of services that can be invoked by an LE/370-conforming High Level Language using the conventional LE/370-defined call interface, and usable by all programs sharing the LE/370 conventions.

Use of these services helps to decrease an application's dependence on the specific form and content of the services delivered by any single operating system.

callable service stub. Contains addressing code to access LE/370 callable service routines.

CICS. Customer Information Control System.

CICS Processing Program Table (PPT). Contains information about CICS load modules (whether the module is in storage or not, its language, use count and entry point address, etc.) needed to complete a transaction.

CICS run-unit. Consists of a statically and/or dynamically bound set of one or more load modules which can be loaded by a CICS loader. A CICS run-unit is equivalent to an LE/370 *enclave.*

CICS translator. A routine that accepts as input an application containing EXEC CICS commands written in an LE/370-conforming HLL and produces as output an equivalent application in which each CICS command has been translated into the language of the source.

CMS. Conversational monitor system.

CMS extended parameter list. A type of parameter list available in the CMS environment consisting of a string composed exactly as the user typed it at the terminal. There is no tokenization performed on the string.

CMS tokenized parameter list. A type of parameter list available in the CMS environment consisting of eight-byte tokens, folded to upper case, terminating with a double word of X'FF'. Not supported under LE/370.

COBOL run unit. A COBOL-specific term which defines the scope of language semantics. This is equivalent to a LE/370 *enclave.*

COMMAREA. A communication area made available to applications running under CICS.

compilation unit. An independently compilable sequence of HLL statements. Each HLL product has different rules for what comprises a compilation unit. Synonymous with *program unit.*

condition. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware/operating system and result in an interrupt. The can also be detected by language-specific generated code or language library code.

condition handler. A user-written or language-specific routine invoked by the LE/370 *condition manager* to respond to conditions.

condition handling. In LE/370, the diagnosis, reporting, and/or tolerating of errors that occur in the run-time environment.

condition manager. Manages conditions in the common execution environment by invoking various user-written and language-specific *condition handlers.*

condition token. In LE/370, a data type consisting of 12 bytes. The condition token contains structured fields that indicate various aspects of a condition including the severity, the associated message number, and information that is specific to a given instance of the condition.

constructed reentrancy. The attribute of applications that contain external data and require additional processing to make them reentrant. Contrast with *natural reentrancy.*

conversational monitor system (CMS). A virtual machine operating system that provides general interactive time sharing, problem solving, and program development capabilities, and operates only under the control of the VM/370 control program.

CPPL (command processor parameter list). The format of a TSO parameter list. When a TSO terminal monitor application attaches a command processor, register 1 contains a pointer to the CPPL, containing addresses required by the command processor.

Customer Information Control System (CICS). A transaction-oriented data base/data communication (DB/DC) system.

D

data set. Under MVS, a named collection of related data records that is stored and retrieved by an assigned name. Equivalent to a CMS *file.*

data type. The properties and internal representation that characterize data.

DBCS. Double byte character set.

DB2. An IBM relational database product.

DD statement. In MVS, serves as the connection between the logical name of a file with the physical name of file.

LE/370 Glossary

A

ABEND. Abnormal end of application.

active routine. The currently executing routine.

AD/Cycle. IBM's framework for Systems Application Architecture (SAA) application development. This framework consists of an integrated set of application development offerings that assist developers throughout the application development life cycle, from enterprise modeling, to code generation, to ongoing maintenance.

AD/Cycle CODE/370. IBM SAA AD/Cycle CoOperative Development Environment/370 Cooperative edit, compile, and debug tool. Consists of a language-sensitive editor and debug tool that operates in a windowed environment on the IBM Operating System/2 (OS/2).

additional heap. An LE/370 heap created and controlled by a call to CEECRHP. See also *below heap, anywhere heap,* and *initial heap.*

address space. Domain of addresses that are accessible by an application.

aggregate. A structured collection of data items that form a single data type. Contrast with *scalar.*

American National Standard Code for Information Interchange (ASCII). The code developed by the American National Standards Institute (ANSI) for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

anywhere heap. The LE/370 heap controlled by the ANYHEAP run-time option. It contains library data, such as LE/370 control blocks and data structures not normally accessible from user code. The anywhere heap may reside above 16M. See also *below heap, additional heap, initial heap.*

application. A collection of one or more routines cooperating to achieve particular objectives.

Application Interface Block (AIB). IMS interface between an application and an IMS database.

argument. An expression used at the point of a call to specify a data item or aggregate to be passed to the called routine.

array. An aggregate that consists of data objects each of which may be uniquely referenced by subscripting.

array element. A data item in an array.

ASCII. American National Standard Code for Information Interchange.

Assembler H. An IBM licensed program. Translates symbolic assembler language into binary machine language.

automatic call library. Contains load modules or object modules that are to be used as secondary input to the linkage editor to resolve external symbols left undefined after all the primary input has been processed.

The automatic call library may be:

- Libraries containing object modules, with or without linkage editor control statements
- Libraries containing load modules
- The library containing LE/370 run-time routines (SCEELKED)

automatic data. Data that does not persist across calls to other routines. Automatic data may be automatically initialized to a certain value upon entry and reentry to a routine.

automatic storage. Storage that is allocated on entry to a routine or block and is freed on the subsequent return. Sometimes referred to as *stack storage* or *dynamic storage.*

B

batch. Pertaining to activity involving little or no user action. Contrast with *interactive.*

below heap. The LE/370 heap controlled by the BELOWHEAP run-time options and containing library data, such as LE/370 control block and data structures on normally accessible from user code. Below heap always resides below 16M. See also *anywhere heap, initial heap, additional heap.*

buffer. An area of storage into which data is read or from which it is written.

by content. See *pass by content.*

by reference. See *pass by reference.*

by value. See *pass by value.*

© Copyright IBM Corp. 1991

Interactive System Productivity Facility/Program Development Facility for MVS, SC34-2089

ISPF Dialog Management Services: Programming Guide, SR20-8686

MVS/ESA*

MVS/Enterprise Systems Architecture Installation: System Generation, GC28-1825

MVS/Enterprise Systems Architecture System Programming Library: Initialization and Tuning, GC28-1828

SAA

Systems Application Architecture: AD/Cycle Concepts, GC26-4531

IBM SAA Common Programming Interface C Reference—Level 2, SC09-1308

Systems Application Architecture Common Programming Interface COBOL Reference, SC26-4354

Systems Application Architecture Common Programming Interface C Reference, SC26-4353

TSO

OS/VS TSO Terminal Monitor Program and Service Routines Logic, SY28-0650

OS/VS TSO Guide to Writing a Terminal Monitor Program or a Command Processor, GC28-0648

OS/VS TSO Command Language Reference, GC28-0646

TSO Messages, GC28-1310

TSO Extensions User's Guide, SC28-1333

VM/ESA

VM/ESA CMS User's Guide, SC24-5460

VM/ESA CMS Command Reference, SC24-5461

VM/ESA XEDIT User's Guide, SC24-5463

VM/ESA XEDIT Command and Macro Reference, SC24-5464

VM/ESA CP Command and Utility Reference, SC24-5519

VM/ESA Installation, SC24-5526

VM/ESA Service Guide, SC24-5527

VS COBOL II

VS COBOL II Application Programming: Language Reference, SC26-4047

VS COBOL II Application Programming Guide Release 3.1, SC26-4045

You can order these publications from Mechanicsburg through your IBM representative.

Bibliography

LE/370 Publications

Language Environment/370 Fact Sheet, GC26-4785

Language Environment/370 Concepts Guide, GC26-4786

Language Environment/370 Programming Guide, SC26-4818

Language Environment/370 Planning for Installation and Customization, SC26-4817

Language Environment/370 Debugging and Run-time Messages Guide, SC26-4829

Language Environment/370 Diagnosis Guide, LY37-3711

Language Environment/370 Licensed Program Specifications, GC26-4774

High Level Language Publications

IBM SAA AD/Cycle COBOL/370

IBM SAA AD/Cycle COBOL/370 General Information, GC26-4762

IBM SAA AD/Cycle COBOL/370 Programming Guide, SC26-4767

IBM SAA AD/Cycle COBOL/370 Planning for Installation and Customization, SC26-4766

IBM SAA AD/Cycle COBOL/370 Diagnosis Guide, LY26-9596

IBM SAA AD/Cycle COBOL/370 Migration Guide, GC26-4764

IBM SAA AD/Cycle COBOL/370 Licensed Program Specifications, GC26-4761

IBM SAA AD/Cycle C/370

IBM SAA AD/Cycle C/370 General Information Manual, GC09-1358

IBM SAA AD/Cycle C/370 Programming Guide, SC09-1356

IBM SAA AD/Cycle C/370 Reference Summary, SX09-1247

IBM SAA AD/Cycle C/370 Diagnosis Guide, LY09-1806

IBM SAA AD/Cycle C/370 Migration Guide, SC09-1359

IBM SAA AD/Cycle C/370 Licensed Program Specifications, GC09-1357

IBM SAA AD/Cycle CoOperative Development Environment/370

IBM SAA AD/Cycle CoOperative Development Environment Fact Sheet, G520-8054

IBM SAA AD/Cycle CoOperative Development Environment General Information Manual, GC26-4661

IBM SAA AD/Cycle CoOperative Development Environment Using the Debug Tool, SC26-4662

IBM SAA AD/Cycle CoOperative Development Environment Debug Tool Reference, SC26-4664

IBM SAA AD/Cycle CoOperative Development Environment Installing AD/Cycle CODE, SC26-4663

IBM SAA AD/Cycle CoOperative Development Environment Licensed Program Specifications, GC26-4665

Related Publications

CICS/ESA

CICS/ESA Installation Guide, SC33-0663

CICS/ESA Operations Guide, SC33-0668

CICS/ESA Problem Determination Guide, SC33-0678

CICS/ESA Definition Manual, SC33-0667

CICS/ESA Application Programming Guide, SC33-0675

C/370 (Pre LE/370 Conforming Version)

C/370 User's Guide Version 2, SC09-1264

DB2

IBM Database 2 Application Programming and SQL Guide, SC26-4377

DFSORT/CMS

DFSORT/CMS User's Guide, SC26-4361

IMS

IMS/ESA Version 3 Release 1 Application Programming Guide: DL/I Calls, SC26-4274

IMS/ESA Version 3 Release 1 Application Programming Guide: Exec DLI Commands, SC26-4280

IMS/VS Version 2 Application Programming Guide, SC26-4178

ISPF

© Copyright IBM Corp. 1991

Appendix H. LE/370 Macros

The macros identified in this appendix are provided as programming interfaces by LE/370. All macros listed here are provided as General-Use Programming Interfaces.

Warning: Do not use as programming interfaces any LE/370 macros other than those identified in this appendix.

- `__sysplist` (see "C/370 Parameter Passing Styles" on page 14)
- `__csplist` (see "C/370 Parameter Passing Styles" on page 14)
- `__pcblist` (see "C/370 Parameter Passing Styles" on page 14)
- `__osplist` (see "C/370 Parameter Passing Styles" on page 14)
- CEENTRY (see "CEEENTRY Macro — Generate an LE/370-conforming Prolog" on page 160)
- CEETERM (see "CEETERM Macro — Terminate an LE/370-conforming Routine" on page 161)
- CEEDSA (see "CEEDSA Macro — Generate a DSA Mapping" on page 162)
- CEECAA (see "CEECAA Macro — Generate a CAA Mapping" on page 162)
- CEEPPA (see "CEEPPA Macro — Generate a PPA" on page 162)
- CEEXPIT (see "Using Macros to Generate the PIPI Table" on page 187)
- CEEXPITY (see "Using Macros to Generate the PIPI Table" on page 187)
- CEEXPITS (see "Using Macros to Generate the PIPI Table" on page 187)
- CEEXOPT (see Chapter 30, "Specifying Run-time Options" on page 206).

© Copyright IBM Corp. 1991

Table 59 (Page 5 of 6). Defaults Currency and Picture Strings based on COUNTRY setting

Country	ID	Default thousands separator	Default currency symbol	Default currency name	Default time picture string	Default date picture string	Default date and time picture string
Union of Soviet Socialist Republics	SU	,	.	Rub	HH:MI:SS	DD mmm. YYYY g.	DD mmm. YYYY g. HH:MI:SS
United Arab Emirates	AE	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
United Kingdom	GB	.	,	5B404040'X	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
United States	US	.	,	$	ZH:MI:SS AP	MM/DD/YY	MM/DD/YY ZH:MI:SS AP
Uruguay	UY	,	.	N$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Vanuatu	VU	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Venezuela	VE	,	.	Bs.	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Western Samoa	WS	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Yemen	YE	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Yugoslavia	YU	,	.	Din	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Zaire	ZR	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Zambia	ZM	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Zimbabwe	ZW	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Table 59 (Page 4 of 6). Defaults Currency and Picture Strings based on COUNTRY setting

Country	ID	Default thousands separator	Default currency symbol	Default currency name	Default time picture string	Default date picture string	Default date and time picture string
Qatar	QA	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Republic of China	TW	.	,	$	HH:MI:SS.999	YY/MM/DD	YY/MM/DD HH:MI:SS.999
Romania	RO	,	.	Lei	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Saint Lucia	LC	.	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Saudi Arabia	SA	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Senegal	SN	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Seychelles	SC	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Sierra Leone	SL	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Singapore	SG	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Somalia	SO	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
South Africa	ZA	.		R	HHhMI:SS.999	YYYY-MM-DD	YYYY-MM-DD HHhMI:SS.999
Spain	ES	,	.	Pts	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Sri Lanka	LK	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Sudan	SD	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Surinam	SR	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Swaziland	SZ	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Sweden	SE	,	.	kr	kl HH.MI.SS	YYYY-MM-DD	YYYY-MM-DD kl HH.MI.SS
Switzerland	CH	,	.	Fr	HH,MI,SS	DD. Mmmmmmmmmmmmmmmmmmmmz YYYY	DD. Mmmmmmmmmmmmmmmmmmmmz YYYY HH,MI,SS
Syria	SY	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Tanzania	TZ	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Thailand	TH	.	,	70404040'X	HH:MI:SS	DD/MM/YYYY	DD/MM/YYYY HH:MI:SS
Togo	TG	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Tunisia	TN	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Turkey	TR	,	.	TL	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Uganda	UG	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Table 59 (Page 3 of 6). Defaults Currency and Picture Strings based on COUNTRY setting

Country	ID	Default thousands separator	Default currency symbol	Default currency name	Default time picture string	Default date picture string	Default date and time picture string
Morocco	MA	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Mozambique	MZ	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Namibia	NA	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Netherlands	NL	;	.	G	HH:MI:SS	DD mmmmmmmmmmmmmmmmmmmmz YYYY	DD mmmmmmmmmmmmmmmmmmmmz YYYY HH:MI:SS
Netherlands Antilles	AN	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
New Caledonia	NC	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
New Zealand	NZ	.	,	$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Nicaragua	NI	.	,	9F404040'X	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Niger	NE	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Nigeria	NG	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Norway	NO	,	.	kr	HH:MI:SS,999	DD.MM.YY	DD.MM.YY HH:MI:SS,999
Oman	OM	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Pakistan	PK	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Panama	PA	.	,	B/	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Papua New Guinea	PG	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Paraguay	PY	,	.	Gs.	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
People's Republic of China	CN	.	,	5B404040'X	HH:MI:SS	YYYY.MM.DD	YYYY.MM.DD HH:MI:SS
Peru	PE	.	,	I/.	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Philippines	PH	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Poland	PL	,	.	E99A4040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Portugal	PT	,	.	Esc.	HH:MI:SS	DD-MM-YYYY	DD-MM-YYYY HH:MI:SS
Puerto Rico	PR	.	,	$	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP

Table 59 (Page 2 of 6). Defaults Currency and Picture Strings based on COUNTRY setting

Country	ID	Default thousands separator	Default currency symbol	Default currency name	Default time picture string	Default date picture string	Default date and time picture string
India	IN	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Indonesia	ID	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Iran	IR	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Iraq	IQ	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Ireland	IE	,	,	5B404040'X	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Israel	IL	.	,	NIS	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Italy	IT	,	.	L.	HH.MI.SS,999	DD/MM/YY	DD/MM/YY HH.MI.SS,999
Jamaica	JM	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Japan	JP	.	,	'5B404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Jordan	JO	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Kenya	KE	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Korea, Republic of	KR	.	,	E0404040'X),	HH:MI:SS	YYYY.MM.DD	YYYY.MM.DD HH:MI:SS
Kuwait	KW	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Lebanon	LB	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Lesotho	LS	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Liberia	LR	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Liecht-enstein	LI	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Libya	LY	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Luxembourg	LU	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Macau	MO	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Madagascar	MG	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mauritania	MR	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mauritius	MU	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Malawi	MW	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Malaysia	MY	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mali	ML	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mexico	MX	.	,	$	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Monaco	MC	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Table 59 (Page 1 of 6). Defaults Currency and Picture Strings based on COUNTRY setting

Country	ID	Default thousands separator	Default currency symbol	Default currency name	Default time picture string	Default date picture string	Default date and time picture string
Costa Rica	CR	,	.	c/.	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Cuba	CU	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Cyprus	CY	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Czecho-slovakia	CS	,	.	D247A240'X),	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Denmark	DK	,	.	kr	HH.MI.SS,99	DD-MM-YY	DD-MM-YY HH.MI.SS,99
Dominican Republic	DO	.	,	$	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Ecuador	EC	,	.	$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Egypt	EG	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
El Salvador	SV	.	,	c/.	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Ethiopia	ET	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Finland	FI	.		Mk	HH.MI.SS,999	DD.MM.YYYY	DD.MM.YYYY HH.MI.SS,99
France	FR	,		F	HH:MI:SS,9	DD.MM.YYYY	DD.MM.YYYY HH:MI:SS,9
Federal Republic of Germany	DE	,	.	DM	HH:MI:SS	DD.MM.YYYY	DD.MM.YYYY HH:MI:SS
Gabon	GA	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Gambia	GM	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Ghana	GH	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Greece	GR	,	.	Drs	HH:MI:SS.999	DD/MM/YY	DD/MM/YY HH:MI:SS.999
Guatemala	GT	.	,	Q	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Guinea-Bissau	GW	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Guyana	GU	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Haiti	HT	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Honduras	HN	.	,	L.	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Hong Kong	HK	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Hungary	HU	,		FT	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Iceland	IS	,	.	kr	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Table 59 (Page 1 of 6). Defaults Currency and Picture Strings based on COUNTRY setting

Country	ID	Default thousands separator	Default currency symbol	Default currency name	Default time picture string	Default date picture string	Default date and time picture string
Afghanistan	AF	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Albania	AL	,	.	Lek	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Algeria	DZ	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Andorra	AD	,	.	'9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Angola	AO	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Antigua	AG	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Argentina	AR	,	.	A.	HH.MI.SS	DD/MM/YY	DD/MM/YY HH.MI.SS
Australia	AU	.	,	$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Austria	AT	,	.	S	HH:MI:SS,999	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS,999
Bahamas	BS	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bahrain	BH	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Bangladesh	BD	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Barbados	BB	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Belgium	BE	,	.	BF	HH:MI:SS,999	DD/MM/YY	DD/MM/YY HH:MI:SS,999
Benin	BJ	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bermuda	BM	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bolivia	BO	,	.	BS	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Botswana	BW	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Brazil	BR	,	.	NCz$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Brunei	BN	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bulgaria	BG	,	.	Lv	HH:MI:SS	YYYY-RRRZ-DD	YYYY-RRRZ-DD HH:MI:SS
Burkina Faso (Upper Volta)	BF	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Burma	BU	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Canada	CA	.	,	$	HH:MI:SS.99	YY-MM-DD	YY-MM-DD HH:MI:SS.99
Cayman Islands	KY	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Chad	TD	,	.	9F404040'X	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Chile	CL	,	.	$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Colombia	CO	,	.	$	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP

Appendix G. IBM-supplied Country Code Defaults

The following table contains the currency symbols and default picture strings for the *country_id* parameter of the COUNTRY run-time option and the *country_code* parameter of the National Language Support callable services. See "COUNTRY" on page 224 and Chapter 37, "National Language Support" on page 338 for more information.

© Copyright IBM Corp. 1991

Appendix F. Running COBOL programs under ISPF

This chapter applies to COBOL users only.

LE/370 supports the following versions of the Interactive System Productivity Facility (ISPF):

- ISPF Version 3 Release 2 for MVS/SP
- ISPF Version 2 Release 2 for VM/ESA

If you code your application using ISPF panels, you can gain interactive access to your COBOL application. When a COBOL application is invoked from ISPF, the parameters that are passed must be compatible with the CMS parameter style supported by LE/370 To do this, specify the CMS option when you use the ISPF command ISPEXEC SELECT PGM to invoke a COBOL routine. Under CMS, both run-time options and program arguments can be passed to the COBOL routine. The order in which these are interpreted by LE/370 depends upon the setting of CBLOPTS (see "CBLOPTS" on page 222).

If you attempt to pass run-time options to a COBOL routine that is invoked from ISPF under MVS, the run-time options will be treated as a program arguments.

Note that COBOL/370 or relinked VS COBOL II NORES applications are now allowed to run concurrently in both screens of the ISPF split screen mode.

For detailed information about using ISPF, see the ISPF publications listed in "Bibliography" on page 477.

© Copyright IBM Corp. 1991

tional dumps and messages are produced as appropriate. The ABEND is then percolated and eventually intercepted by the LE/370 ESTAE exit. Condition handling then continues as described in Chapter 19, "Condition Management" on page 83.

By the time the LE/370 ESTAE exit intercepts the ABEND, the SORT has been terminated. LE/370 moves the current resume cursor to the return point where SORT was invoked and reflects the deletion of stack frame (and associated load modules) following the SORT invocation. Any user condition handlers associated with these stack frames (that is, those following the SORT invocation) do not get control. In addition, user handlers established by the routine that initiated the SORT/MERGE do not get control.

For a detailed description of LE/370 condition handling actions, see Chapter 19, "Condition Management" on page 83.

To transfer sorted records to a file without any further processing, specify:

```
SORT...GIVING
```

User Exit Considerations

User exits (E15 for sort, E35 for merge) are triggered by any invocation of SORT and MERGE in COBOL source code. These exits include any *input* or *output procedures*. However, the exits are not invoked when a COBOL USING or GIVING statement is in effect and the files qualify for FASTSRT.

The exits are treated differently by LE/370 than those triggered by a direct invocation of DFSORT. LE/370 treats user exits triggered by native HLL invocations (such as COBOL SORT) as part of the enclave of the routine that invoked DFSORT. The SVC LINK used to invoke DFSORT is *not* considered by LE/370 to initiate a new implicit nested enclave. This is not the case for direct invocations of DFSORT (see your *DFSORT Application Programming Guide* for more information).

ILC is permitted within the DFSORT user exits as long as ILC is permitted within the same load module. ILC is not permitted in dynamically loaded routines.

Condition Handling Considerations

This section summarizes how LE/370 condition handling behaves when SORT/MERGE is initiated by a COBOL/370 or VS COBOL II routine.

Program Interrupts

User handlers established by the routine that initiated the SORT/MERGE are able to handle program interrupts as they are presented to the condition manager by a condition token. Normal condition handling as described in Chapter 19, "Condition Management" on page 83 will then occur.

HLL-specific handlers and user handlers established while in the input or output procedure are not supported. The results are unpredictable if a user handler is established in an input or output procedure. The condition handler does not attempt to diagnose this case.

HLL-specific handlers and user handlers established by the routine *called* by an input or output procedure are able to handle program interrupts. However, because these exits are typically invoked many times (equivalent to the number of records being sorted for each exit), it is recommended that you register the handler within the application that initiated the SORT/MERGE in order to save overhead.

LE/370-signaled conditions

HLL-specific handlers and user handlers established by the routine that initiated the SORT/MERGE are able to handle any condition signaled by LE/370. Normal condition handling as described in Chapter 19, "Condition Management" on page 83 will then occur.

ABENDs

When there is an ABEND, the DFSORT ESTAE exit intercepts the ABEND. Various cleanups and recoveries are performed by the DFSORT ESTAE. Informa-

Appendix E. Sort/Merge Considerations

Under LE/370, you can invoke the sort facility to sort or merge records in a particular sequence. In a *sort* operation, an unordered sequence of input data is arranged according to a specified key or pattern and placed into an output file. A *merge* operation compares two or more files that have already been sorted according to an identical key and combines them in a specified order in an output file.

There are generally two ways to invoke the sort facility in the common run-time environment:

- "Native" HLL invocations (such as COBOL's SORT and MERGE verbs)
- "Non-native" HLL direct invocations of DFSORT (for example, from assembler routines, JCL, or ISPF).

Under LE/370, your IBM sort/merge licensed program must be DFSORT, or an equivalent that honors the DFSORT extended parameter list. However, whenever DFSORT is mentioned in this chapter, any other equivalent SORT product could be used.

Invoking DFSORT Directly

For information about using the methods described above to invoke DFSORT directly, see your *DFSORT Application Programming Guide*.

Using the COBOL SORT and MERGE Verbs

This section contains a high-level overview of SORT and MERGE. It is designed to introduce you to concepts that help you understand some of the special considerations for using these COBOL statements in the common run-time environment. For a detailed description of how to use SORT and MERGE, see your *IBM SAA AD/Cycle COBOL/370 Programming Guide*.

A COBOL routine that contains a sort operation can be organized so that one or more input files are read and operated on by an *input procedure* before the files are actually sorted, for example:

```
SORT...INPUT PROCEDURE
```

You can also specify an *output procedure* that processes the files after they are sorted:

```
SORT...OUTPUT PROCEDURE
```

These input and output procedures can be used to add, delete, alter, edit, or otherwise modify the records.

You can also sort records under COBOL without any processing by the input and output procedures. For example, to read records into a new file for sorting without any preliminary processing, specify:

```
SORT...USING
```


© Copyright IBM Corp. 1991

Example of JCL

An example of the run-time JCL to run an application that uses MTF is shown in Figure 114. This figure shows the JCL that is unique to running MTF, as well as the other JCL the application would typically require. (Some applications might require additional DD statements.)

```
//GO        EXEC PGM=MTASKPGM
//STEPLIB   DD   DSN=USERPGM.LOAD,DISP=SHR
//STDIN01   DD   DSN=USERPGM.INPUT,DISP=SHR
//STDOUT02  DD   SYSOUT=S,DCB=(RECFM=F)
```

Figure 114. Example Run-Time JCL

MTASKPGM is the name of the main task program load module, and is the load module that gets control when MVS first starts running the application. In this example, this load module is contained in data set USERPGM.LOAD, which is referred to by the STEPLIB DD statement. USERPGM.LOAD also contains the parallel functions.

The STDIN01 DD statement specifies the data set that contains the application's input data for the first task. The STDOUT02 DD statement specifies that printed output aside from run-time error messages from the second subtask is to be written to SYSOUT class S and that the record format is to be fixed-length. These DD statements are necessary only if you do not want to accept the defaults.

Running Applications That Use MTF

To run your application, you use the usual MVS JCL for C/370 applications, plus a few additional JCL statements that are required to run MTF.

STEPLIB DD Statement

You must ensure that the library containing the load module(s) is specified on the STEPLIB DD statement in your JCL, in addition to the other libraries normally specified

```
//STEPLIB    DD     DSN=user.dsn,DISP=SHR
```

where:

user.dsn
: is the name of the load module library that contains the parallel functions load module.

  The parallel functions load module (*parallel_loadmod_name*), specified on the call to `tinit`, must be contained in this data set.

  When running under TSO, because of restrictions regarding the STEPLIB DD statement, you must allocate the ddname EDCMTF to the user.dsn data set as well as adding user.dsn to the STEPLIB concatenation list.

DD Statements for Standard Streams

For standard streams, MTF assigns a unique object-time output file to each parallel function. These output files contain diagnostic messages that the library can issue while the parallel functions are running. They also contain output directed to the standard streams (`stderr` and `stdout`) by parallel functions and input from the standard stream `stdin`.

Because these files are automatically allocated while the application is running, you need not supply DD statements for them unless you wish to override the default device type or other file characteristics. The default device type is a terminal in TSO or SYSOUT=* in batch.

If you do supply DD statements, use the following ddnames:

- `stdin`stn for files containing input for operations such as `getc`
- `stderr`stn for files containing diagnostic messages
- `stdout`stn for files containing output from operations such as `printf`.

where *stn* is the 2-digit subtask number; that is, 01, 02, 03, and so on. Thus, for example, if you had four subtasks and the first two used `printf` functions, you would use the ddnames `stdout01`, `stdout02`, `stderr01`, `stderr02`, `stderr03`, and `stderr04`.

For example, the following JCL sequence uses the standard C/370 cataloged procedure EDCCL to compile and link-edit the C source for the parallel functions (stored in data set USERPGM.C(SUBTASK)) and create a parallel functions load module named SUBTASK in data set USERPGM.LOAD. This load module contains the module EDCMTFS, and has EDCMTFS as the load module's entry point.

```
//SUBTASK      EXEC  EDCCL,
//                   INFILE='USERPGM.C(SUBTASK)',
//                   OUTFILE='USERPGM.LOAD(SUBTASK),DISP=OLD'
//LKED.SYSLIN   DD
//              DD *
  INCLUDE SYSLIB(EDCMTFS)
  ENTRY   CEESTART
/*
```

Figure 112. Sample JCL to Compile and Link Parallel Functions

Link-Editing Considerations

Do not specify the NE linkage-editor option when link-editing the parallel functions load module. MTF cannot schedule parallel functions that are contained in a load module link-edited with the NE option.

Modifying Run-Time Options

You can alter the #pragma runopts options ISASIZE, ISAINC, and HEAP within the EDCMTFS module for each subtask but you must re-compile the module under the same name.

```
/*******************************************************************/
/* Modify the isa/isainc/heap subparameters in the following line   */
/* as required to meet your needs. Ensure that your version (compiled*/
/* and linked) is then accessed in your link-edit of the parallel    */
/* module in place of the pre-built EDCMTFS found in SEDCBASE.       */
/*******************************************************************/
#pragma runopts(ISA(8K),ISAINC(4K),HEAP(4K,4K,ANY,FREE))
/*******************************************************************/
/* The following lines must remain unmodified to ensure proper       */
/* operation of MTF.                                                 */
/*******************************************************************/
#pragma runopts(STAE,SPIE,NOREPORT,NOTEST,ARGPARSE,REDIR,NOEXECOPS)
int main(int argc, char **argv) { return tsetsubt(argc,argv); }
```

Figure 113. Source code for EDCMTFS

Appendix D. Using the C/370 Multitasking Facility

Compiling and Linking Applications That Use MTF

Applications that use MTF run using two MVS load modules: a load module that contains the main task program, and a load module that contains the parallel functions. You compile and link-edit these two load modules in the same procedure as non-MTF C applications.

Creating the Main Task Program Load Module

The main task program load module is the load module that first receives control when MVS starts running your application. It is the load module named in the PGM keyword of the EXEC statement. This load module contains your application's C main function plus all other functions that are to run as part of the main task. The MTF functions can be invoked from any of the C functions contained in the main task load module and do not necessarily have to be invoked from the C function called main.

The procedures that you normally use to compile and link-edit a C/370 application can be used to create the main task program load module. For example, the following JCL sequence (Figure 111 on page 461) uses the standard C/370 cataloged procedure EDCCL to compile and link-edit the C source for the main task program (stored in data set USERPGM.C(MTASKPGM)) and create a main task program load module named MTASKPGM in data set USERPGM.LOAD.

```
//MTASKPGM   EXEC  EDCCL,
//           INFILE='USERPGM.C(MTASKPGM)',
//           OUTFILE='USERPGM.LOAD(MTASKPGM),DISP=OLD'
```

Figure 111. Sample JCL to Compile and Link Main Task Program

Creating the Parallel Functions Load Module

The parallel functions load module is the load module named in the call to the MTF library function tinit. This single load module contains all of your main task program's parallel functions. It must not contain any user's C main routines. C/370 itself provides the EDCMTFS module to act as the C main function in the parallel module. EDCMTFS controls processing of the parallel functions as they are scheduled (using tsched calls) to the subtasks. The source code for the EDCMTFS module is included in Figure 113 on page 462.

The procedures that you normally use to compile and link-edit a C/370 application must be modified so that the library module CEESTART will be the entry point of the parallel functions load module.

When link-editing this load module, you must include the following linkage editor control statements:

```
INCLUDE SYSLIB(EDCMTFS)
ENTRY   CEESTART
```


© Copyright IBM Corp. 1991

according to the following hierarchy:

(P) indicates that the timestamp is extracted from the object module from the date form of #pragma comment, or from the timestamp form of #pragma comment, whatever comes first.
(D) indicates that the timestamp is based on the time that the C/370 Object Library Utility DIR command was last issued.
(F) indicates that the timestamp is the date of the object module file at the time the ADD or GEN command was issued for the member. This is applicable to VM only.
(T) indicates that the timestamp is the time that the ADD command was issued for the member. This is applicable to MVS only.

[3] User Comments
The user form of #pragma comment are displayed. These comments are extracted from the END record. It is possible to manually add such comments on multiple END records and have them displayed in the listing. See the *IBM SAA AD/Cycle C/370 Programming Guide* on #pragma comment for more information on their format on the END record.

[4] Symbol Information
Immediately following the Member Heading (and user comments, if any) is a list of the defined objects contained within that member. Each symbol is prefixed by Type information enclosed in parenthesis and either External Name or Function Name. Function Name will appear provided the object module was compiled with the LONGNAME option and the symbol is the name of a defined external function. In all other cases External Name will be displayed. The Type field gives additional information on each symbol. That is:

'L' indicates that the name is a long name
'S' indicates that the name is a short name
'W' indicates that this is a writable static object. If no 'W' is present, then this is not a writable static object.

```
   (L) External Name: Longtypedefint
   (L) External Name: longTypedefint
   (L) External Name: longtype
   (L) External Name: longtyped
   (L) External Name: chartypedef
   (L) External Name: Chartypedef
   (L) External Name: chartypedef_name_of_255_characters______01234567
                      89012345678901234567890123456789012345678901234 5
                      67890123456789012345678901234567890123456789 0123
                      45678901234567890123456789012345678901234567890 1
                      23456789012345678901234567890123456789012345678 9
                      0123456789012 3
   (L) External Name: Longname_int
   (L) External Name: Longname_double
   (L) External Name: Longname_float
   (L) External Name: Longname_char
   (L) External Name: longname_int
   (L) External Name: longname_double
   (L) External Name: longname_float
   (L) External Name: longname_char
   (L) External Name: Longname_float_temp
   (L) External Name: longname_char1

*---------------------------------------------------------------*
*   Member Name: C144004                    (T) ??/??/?? ??:??:?? *  [2]
*                                               ????????  R?? M?? *
*---------------------------------------------------------------*

  User Comment: this is program C144004 C                           [3]

   (L) Function Name: func_in_                                      [4]
   (L) Function Name: func_in_PROGRAM_C144004
   (L) Function Name: func_in_PROGRAM_C144004_which_calls_a_func_in_PR
                      OGRAM_C144003
   (L) Function Name: func_in_PROGRAM_C144004_which_calls_a_func_in_PR
                      OGRAM_C144004
  (WL) External Name: Longname_int1
  (WL) External Name: Longname_Float
  (WL) External Name: Longname_Float1
  (WL) External Name: LongnamE_double
  (WL) External Name: Longname_double_e
  (WL) External Name: LonGname_char
  (WL) External Name: longname_Int
  (WL) External Name: lOngname_float
  (WL) External Name: longname_Double
  (WL) External Name: longnamE_char

======== E N D   O F   O B J E C T   L I B R A R Y   M A P =========
```

[1] Map Heading

The heading contains the product number, the compiler release number, the compiler version number, the date and the time the C/370 Object Library Utility step commenced. The name of the library immediately follows the heading. To the right of the name of the library is the start time of the last C/370 Object Library Utility step that updated the C370LIB-directory.

[2] Member Heading

The name of the object module member is immediately followed by the ID of the processor that produced the object module. The processor ID is based on the presence of an END record in the object module that has the processor information in the appropriate format. If this information is not present, the Processor ID field is not listed.

The Timestamp field is presented in *yy/mm/dd* format. The meaning of the timestamp is enclosed in parenthesis. That is, the C/370 Object Library Utility retains a timestamp for each member and selects the time

```
=====================================================================
|                     Object Library Utility Map                    |
|                                                                   |  [1]
| C370LIB:???????? V? R? M?? IBM AD/Cycle C/370     ??/??/?? ??:??:?? |
=====================================================================

 Library Name: ???????.LONGNAME.C370LIB             ??/??/?? ??:??:??

*-------------------------------------------------------------------*
*  Member Name: C144003                        (P) ??/??/?? ??:??:?? *  [2]
*                                                  ????????  R?? M?? *
*-------------------------------------------------------------------*

  User Comment: this is program C144003 C                               [3]

     (L) Function Name: sfail                                           [4]
     (L) Function Name: fail
     (L) Function Name: pass
     (L) Function Name: term
     (L) Function Name: check
     (L) Function Name: func_in_PROGRAM_C144003
     (L) Function Name: func_in_PROGRAM_C144003_which_calls_a_func_in_PR
                        OGRAM_C144003
     (L) Function Name: func_in_PROGRAM_C144003_which_calls_a_func_in_PR
                        OGRAM_C144004
     (L) Function Name: func_in_PROGRAM_C144003_which_calls_a_func_in_PR
                        OGRAM_C144004_which_calls_a_func_in_PROGRAM_C144
                        003
     (L) Function Name: func_in_PROGRAM_C144003_which_calls_a_func_in_PR
                        OGRAM_C144004_which_calls_a_func_in_PROGRAM_C144
                        004_Also_this_func_has_a_255_character_function_
                        name_0123456789012345678901234567890123456789012
                        3456789012345678901234567890123456789012345678890
                        123456789012345
     (L) Function Name: main
     (L) External Name: __vsoft
     (L) External Name: longstructure_name_of_255_characters012345678901
                        234567890123456789012345678901234567890123456789
                        012345678901234567890123456789012345678901234567
                        890123456789012345678901234567890123456789012345
                        678901234567890123456789012345678901234567890123
                        45678901234567A
     (L) External Name: longstructure
     (L) External Name: Longstructure
     (L) External Name: longstru
     (L) External Name: LoNgStRuC
     (L) External Name: longstructure_name_of_255_characters012345678901
                        234567890123456789012345678901234567890123456789
                        012345678901234567890123456789012345678901234567
                        890123456789012345678901234567890123456789012345
                        678901234567890123456789012345678901234567890123
                        45678901234567a
     (L) External Name: longunio
     (L) External Name: longunion
     (L) External Name: LoNgUnIo
     (L) External Name: LoNgUnIoN
     (L) External Name: longunion_name_of_255_characters0123456789012345
                        678901234567890123456789012345678901234567890123
                        456789012345678901234567890123456789012345678901
                        234567890123456789012345678901234567890123456789
                        012345678901234567890123456789012345678901234567
                        89012345678901X
     (L) External Name: longunion_name_of_255_characters0123456789012345
                        678901234567890123456789012345678901234567890123
                        456789012345678901234567890123456789012345678901
                        234567890123456789012345678901234567890123456789
                        012345678901234567890123456789012345678901234567
                        89012345678901x
```

If you do not specify a data set name, a name is automatically generated using the library name and the qualifier MAP. For example, if the input library data set is called TEST.OBJ and your user prefix is FRANK, the data set name generated for the map is FRANK.TEST.OBJ.MAP.

Usage Note

1. Under TSO, you can use the C370LIB CLIST or the CC CLIST while specifying the C370LIB parameter. The C370LIB parameter of the CC CLIST specifies that if the object module from the compile is directed to a member of a PDS, the C370LIB-directory is to be updated.

   For more information about the CC CLIST, see the *IBM SAA AD/Cycle C/370 Programming Guide*.

Example

In the following example, the C/370 program WALTER.SOURCE(SUB1) is compiled for L-names. WALTER is the user prefix and the object module is placed in the data set WALTER.SOURCE.OBJ(SUB1).

Because the C370LIB parameter of CC was specified, the C370LIB-directory of WALTER.SOURCE.OBJ is updated. The member SUB2 is deleted (in the second line of the example) from the object module library WALTER.SOURCE.OBJ using the DEL command of C370LIB, the C/370 Object Library Utility. The C370LIB-directory for the library is updated in the process.

```
CC SOURCE(SUB1) COPT('LO') C370LIB
C370LIB DEL LIB(SOURCE.OBJ(SUB2))
```

C/370 Object Library Utility Map

The C/370 Object Library Utility produces a listing for a given library when the MAP command is specified. The listing contains information on each member of the library. In the sample map, the bracketed numbers refer to the comments that follow.

If you request a map for the library WALTER.SOURCE.OBJ, use:

```
//OBJLIB EXEC EDCLIB,OPARM='MAP',LIBRARY='WALTER.SOURCE.OBJ'
```

Creating an Object Library under TSO

The C370LIB CLIST can be used to create an object library.

Syntax

```
▸▸──C370LIB──────────────────────────────────────────────────────────▸

▸──┬──ADD──LIB──(─libname─(─membername─)─)──┬────────────────────┬──┬──▸◂
   │                                        └──OBJ──(─objname─)──┘  │
   ├──DEL──LIB──(─libname─(─membername─)─)──────────────────────────┤
   ├──MAP──LIB──(─libname─)──┬─────────────────┬────────────────────┤
   │                         └──LIST─(─map─)───┘                    │
   └──DIR──LIB──(─libname─)─────────────────────────────────────────┘
```

ADD

adds (or replaces) an object module to an object library.

If the ADD function is used to insert an object module into a member of a library that already exists, the previous member is deleted prior to the insert. However, if the source data set is the same as the target data set, the member is not deleted and only the C370LIB-directory is updated as appropriate.

DEL

deletes an object module from an object library.

MAP

lists the names (entry points) of object library members.

DIR

builds the .C370LIB-directory. The C370LIB-directory contains the names (entry points) of library members.

The DIR function is necessary only if object modules were previously added or deleted from the library without using C370LIB.

LIB *(libname(membername))*

specifies the target data set for the ADD and DEL functions. The data set name must contain a member specification to indicate what member is to be created, replaced or deleted.

OBJ(*objname***)**

specifies the source data set containing the object module that is to be added to the library. If you do not specify a data set name the target data set specified in LIB(libname(membername)) is used as the source.

LIB(*libname***)**

specifies the object library for producing a map or for building a C370LIB-directory to be built.

LIST(*map***)**

specifies the data set that is to contain the library map.

If an asterisk (*) is specified, the library map is directed to your terminal.

membername
specifies the name(s) of TXTLIB member(s) that you want to delete.

FILENAME
indicates that all the filenames specified (fn1 ...) are used as the member names for their respective entries in the TXTLIB file.

Usage Notes

1. C370LIB must be used to update a TXTLIB containing TEXT files that are produced by compiling C programs with the LONGNAME option. The VM TXTLIB command cannot be used to do this directly. An error may result if you attempt to do this.
2. When a TEXT file is added to a library, its member name is selected according to the following hierarchy (from most to least important):
   - from the filename, if the FILENAME option is specified
   - from the NAME control statement, if present, in the TEXT file
   - from the filename, otherwise
3. The CMS TXTLIB command GEN, ADD and DEL functions will be used as part of the C370LIB GEN, ADD and DEL functions. Therefore, unless otherwise stated, any TXTLIB restrictions apply also to C370LIB. See the appropriate manual listed in "Bibliography" on page 477 for more information on the TXTLIB command.
4. Members must be deleted by their member name. Any attempt to delete using a name other than the member name results in a warning message.

Example

In the following example, the C programs SUB1 C and SUB2 C are compiled for L-names. The function library, SUBLIB TXTLIB A, is created with SUB1 TEXT using the GEN command of C370LIB, the C/370 Object Library Utility. SUB2 TEXT is added to the library using the ADD command.

```
CC SUB1 (LO
CC SUB2 (LO
C370LIB GEN SUBLIB SUB1
C370LIB ADD SUBLIB SUB2
```

Creating an Object Library under MVS Batch

Under MVS batch, the following cataloged procedures include a C/370 Object Library Utility step:

EDCLIB	Maintain an object library
EDCCLIB	Compile and maintain an object library.

EDCLIB is shipped with LE; EDCCLIB, with the compiler.

You can also write JCL to include an Object Library Utility step. For example, you can use the following JCL to compile the C program WALTER.SOURCE(SUB1) for L-names and add to WALTER.SOURCE.OBJ(SUB1). The C370LIB-directory for the library, WALTER.SOURCE.OBJ, is updated in the process.

```
//COMPILE EXEC EDCCLIB,INFILE='WALTER.SOURCE(SUB1)',CPARM='LO',
//        LIBRARY='WALTER.SOURCE.OBJ',MEMBER='SUB1'
```

Syntax

```
▸▸──C370LIB──┬─────────────────────────────────────────────┬──▸◂
             │                ┌──────┐                      │
             │                ▼      │                      │
             ├─GEN──libname────fn────┴──┬────────────────┬──┤
             │                          └─(──FILENAME────┘  │
             │                ┌──────┐                      │
             │                ▼      │                      │
             ├─ADD──libname────fn────┴──┬────────────────┬──┤
             │                          └─(──FILENAME────┘  │
             │                ┌──────────────┐              │
             │                ▼              │              │
             ├─DEL──libname────membername────┴──────────────┤
             ├─MAP──libname─────────────────────────────────┤
             └─DIR──libname─────────────────────────────────┘
```

GEN
creates a TXTLIB on your A disk. If a TXTLIB with the same name already exists, it is replaced.

ADD
adds TEXT files to the end of an existing TXTLIB on a read/write disk. No checking is done for duplicate names, entry points, or CSECTs.

DEL
deletes members from a TXTLIB on a read/write disk and compresses the TXTLIB to remove unused space. If more than one member exists with the same name, only the first entry is deleted.

MAP
lists the names (entry points) of TXTLIB members.

MAP produces a file, libname MAP, on your A disk. See "C/370 Object Library Utility Map" on page 457 for more information on the map.

DIR
builds the TXTLIB C370LIB-directory. The *C370LIB-directory* contains the names (entry points) of library members.

The DIR command can be used if the library was accidentally updated without the C/370 Object Library Utility or to build the C370LIB-directory for a library of object modules containing S-names.

The DIR function is *necessary* only if TEXT files were previously added or deleted from the TXTLIB without using C370LIB.

libname
specifies the filename of a file (filetype of TXTLIB) which is to be created or listed, or from which members are to be added or deleted, or for which a C370LIB-directory is to be built.

fn1
specifies the name(s) of file(s) with filetype(s) of TEXT, that you want to add to a TXTLIB.

Appendix C. Using the C/370 Object Library Utility

The C/370 Object Library Utility is used to update libraries of object modules. A library on VM is a text library (TXTLIB) with object modules as members. On MVS, a library is partitioned data set (PDS) with object modules as members.

You can include object modules from other languages using the C/370 Object Library Utility. Object libraries provide for convenient packaging of object modules. With the C/370 Object Library Utility, a library can contain object modules with L-names and object modules with S-names. See Chapter 7, "Making Your Application Reentrant" on page 28 and the *IBM SAA AD/Cycle C/370 Programming Guide* for more information on L-names and S-names.

The C/370 Object Library Utility is used to create information for libraries of object modules containing L-names or S-names. This information is stored in a member of the library and will be referred to as the C370LIB-directory. This special member, created by the C/370 Object Library Utility, contains information on each member of the library such as the defined L-names or defined S-names that the member contains.

Notes:

1. The TXTLIB command under VM also creates object libraries but it does not allow you to include external names greater than 8 characters long. The syntax for the C/370 Object Library Utility is similar to the TXTLIB command on VM modules.
2. Because C/370 will generate private code if you do not include a pragma csect(code) directive in your source or if you do not create a NAME control statement using the ALIAS compile-time option, you should use the FILENAME option on the TXTLIB or C370LIB commands. For more information about pragma csect(code), see the *IBM SAA AD/Cycle C/370 Programming Guide.*

Commands to add object modules to a library, to delete object modules from a library, or to build the C370LIB-directory for a library are described in "Creating an Object Library under VM" on page 453. The MAP command is provided to list the contents of the C370LIB-directory.

Creating an Object Library under VM

Use either the CC (see the *IBM SAA AD/Cycle C/370 Programming Guide* for more information) or C370LIB EXECs to create an object library.

© Copyright IBM Corp. 1991

Table 58 (Page 1 of 2). Summary of Types

Type of Application	**How It Is Called**	**Module Entry Point**	**Datasets Required at Execution Time**	**Run-time options and Other Considerations**
A mainline function that requires C/370 library functions	From the command line, JCL, or an EXEC or CLIST	`EDCXSTRL` must be explicitly included at bind time.	C/370 library functions	Run-time options are specified by `#pragma runopts` in the compilation unit for the `main()` function. The `SPIE`, HEAP and STACK options are honored, except that the stack will default to above the 16M line.
A mainline function that uses storage preallocated by the caller.	From Assembler code.		C/370 library functions are optional; the caller must load these functions and pass their addresses to `EDCXSTRX`, if required to by the application.	Run-time options are specified by `#pragma runopts` in the `main()` function. The `SPIE` option is honored if C/370 library functions are required.
An exit	From (typically) Assembler code, with a structured parameter list.		C/370 library functions, if required	Run-time options are specified by `#pragma runopts` in the compile unit for the entry point. The HEAP and STACK options are honored, except that the stack will default to be above the 16M line. The `SPIE` option is honored if C/870 library functions are called for.
A C subroutine called from Assembler language using a pre-established persistent environment.	A *handle*, the address of the subroutine, and a parameter list are passed to `EDCXHOTU`.		C/370 library functions are optional, depending on the way the *handle* was set up.	Run-time options are specified by `#pragma runopts` in any compile unit. The HEAP and STACK options are honored, except that the stack will default to above the 16M line. The `SPIE` option is honored if C/370 library functions are called for. The runopts in the first object module in the link-edit that contains runopts will prevail, even if this compilation unit is part of the calling application. The environment is established by calling `EDCXHOTC` or `EDCXHOTL` (if library functions are required). These functions return a value (the *handle*) which is used to call functions that use the environment.
A server	User code includes a stub routine that calls `EDCXSRVI`. This causes the server to be loaded and control to be passed to its entry point.	`EDCXSTRT`, or `EDCXSTRL`, depending on whether the server needs C/370 library functions.	C/370 library functions, if required by the server code.	Run-time options are the same as for `EDCXSTRL` or `EDCXSTRT`. The author of the server must supply stub routines which call `EDCXSRVI` and `EDCXSRVN` to initialize and communicate with the server. These are bound with the user application.
A user of an application server			The server and C/370 library functions, if required by the server.	The author of the server must supply stub routines which call `EDCXSRVI` and `EDCXSRVN` to initialize and communicate with the server.

Building System Exit Routines

There are no special considerations for building System Exit routines. These routines can be linked with their callers or dynamically loaded and invoked. SCEESPC TXTLIB (under VM) or CEE.V1R1M0.SCEESPC (under MVS) must be available at link-edit. If C library functions are required by the exit routines, the libraries SCEELKED (under VM) or CEE.V1R1M0.SCEELKED (under MVS) must also be made available *after* SCEESPC or EDC.V1R1M0.SCEESPC. If the routines were compiled with OPT(2), the entry point must be explicitly named in the link-edit input (under MVS), or identified using the RESET option on the LOAD command (under VM).

Note: You must compile these programs with the NOSTART option.

Building Persistent C Environments

There are no special considerations for building applications that use persistent C environments. Under VM, the LIBE option of the LOAD command will cause the proper object modules to be included from SCEESPC TXTLIB. Under MVS, the data set EDC.V1R1M0.SCEESPC contains the object modules to be included.

If C/370 Library functions are required by any routine called in this environment, the library stub routines should also be made available at link time *after* SCEESPC or CEE.V1R1M0.SCEESPC.

Note: You must compile these programs with the NOSTART option.

Building User-Server Environments

To build your server application, follow the rules for building a freestanding application as described in "Building Freestanding Applications under VM" on page 446 and "Building Freestanding Applications under MVS" on page 448.

There are no special considerations for building user applications. Under VM, the LIBE option of the load command causes the proper object modules to be included from SCEESPC TXTLIB. The automatic call facility causes the right routines from the TXTLIB (under VM, using the LIBE option) or SYSLIB (under MVS) to be included.

Note: You must compile servers with the NOSTART option.

Summary

Table 58 (Page 1 of 2). Summary of Types

Type of Application	How It Is Called	Module Entry Point	Datasets Required at Execution Time	Run-time options and Other Considerations
A mainline function that requires no C/370-specific library functions	From the command line, JCL, or an EXEC or CLIST	EDCXSTRT must be explicitly included at bind time	None	Run-time options are specified by #pragma runopts in the compilation unit for the main() function. The HEAP and STACK options are honored. STACK defaults to above the 16M line.

```
//PRLK      EXEC  PGM=EDCPRLK,PARM='MAP,NCAL'                      [110-1]
//STEPLIB   DD    DSN=CEE.V1R1M0.SCEERUN,DISP=SHR
//SYSMSGS   DD    DSN=CEE.V1R1M0.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSOUT    DD    SYSOUT=*
//SYSMOD    DD    DSN=&&OBJ,SPACE=(TRK,(1,1)),UNIT=SYSDA,
//                DCB=(BLKSIZE=400,RECFM=FB,LRECL=80),
//                DISP=(MOD,PASS)
//SYSIN     DD    DSN=RETS321.OBJ,DISP=SHR                        [110-2]
//*
//*
//LKED      EXEC  PGM=IEWL,PARM='MAP,XREF,LIST'
//SYSUT1    DD    SPACE=(CYL,1),UNIT=SYSDA
//PRELINK   DD    DSN=&&OBJ,DISP=(OLD,DELETE)                     [110-3]
//SYSLIN    DD    *
 INCLUDE SYSLIB(EDCXSTRT)                                         [110-4]
 INCLUDE PRELINK                                                  [110-5]
 INCLUDE SYSLIB(EDCXEXIT)                                         [110-6]
 INCLUDE SYSLIB(EDCRCINT)                                         [110-7]
/*
//SYSPRINT  DD    SYSOUT=*
//SYSLMOD   DD    DSN=&&GOSET(GO),
//                UNIT=SYSDA,SPACE=(TRK,(1,1,1)),
//                DISP=(NEW,PASS)
//SYSLIB    DD    DSN=CEE.V1R1M0.SCEESPC,DISP=(SHR,PASS)
//          DD    DSN=CEE.V1R1M0.SCEELKED,DISP=(SHR,PASS)
//GO        EXEC  PGM=*.LKED.SYSLMOD
```

Figure 110. Building and Running a Reentrant Freestanding MVS Routine

Notes

[110-1] The C/370 Prelinkage Utility must be used for modules compiled with the RENT compile-time option.

[110-2] This is the object module created by compiling the sample module with the RENT and NOSTART compile-time options.

[110-3] The output from the prelinker is made available to the linkage editor.

[110-4] The alternate initialization routine (EDCXSTRT in this example) must be included explicitly in the module. If this is not the first CSECT in the module it must be explicitly named as the module entry point.

[110-5] The prelinked output is included in the load module.

[110-6] EDCXEXIT must be explicitly included if the exit() function is used in the application.

[110-7] The routine EDCRCINT must be explicitly included in the module if the RENT compile-time option is used. No error will be detected at load time if this routine is not explicitly included. At execution time, ABEND 2106, reason code 7205, will result if EDCRCINT is required but not included.

```
//ANDREWA JOB   (518B,2326),'CL EXAMPLE FOR STRL',CLASS=C,MSGCLASS=S,
//        NOTIFY=ANDREW,USER=ANDREW, PASSWORD=????????,
//        MSGLEVEL=(1,1),REGION=4M
/*JOBPARM  T=2.5,L=99,P=PROC03
/*ROUTE    PRINT   U844
//*-------------------------------------------------------------
//*****************************************
//**** COMPILE AND LINK FOR STRL ENTRY POINT
//*****************************************
//C106001    EXEC    EDCCL,
//      INFILE='ANDREW.SPC.SOURCE(C106000)',
//      OUTFILE='ANDREW.SPC.LOAD(C106000),DISP=SHR',
//      CPARM='OPT(2),NOSEQ,NOMAR,NOSTART',
//      LPARM='RMODE=ANY,AMODE=31'
//LKED.SYSLIB  DD DSN=CEE.V1R1M0.SCEESPC,DISP=SHR
//             DD DSN=&VSCCHD&CVER&CBASE,DISP=SHR
//             DD DSN=&COMHD&COMVER&COMBASE,DISP=SHR
//LKED.SYSIN   DD *
  INCLUDE SYSLIB(EDCXSTRL)
  ENTRY EDCXSTRL
/*
```

Figure 108. Compile and link using EDCCL

Special Considerations for Reentrant Modules (MVS)

A simple freestanding routine that does not require C/370 library functions is shown in Figure 109 on page 449. This routine uses the `exit()` function, which is normally part of the C/370 library but (like `sprintf()`) is available to freestanding routines without requiring the dynamic library. This routine is not naturally reentrant, but the resulting load module will be reentrant.

```
#include <stdlib.h>

int main() {
    static int i[5]={0,1,2,3,4};
    exit(320+i[1]);
}
```

Figure 109. Sample Reentrant Freestanding VM Routine

The JCL required to build and execute this routine is shown in Figure 110 on page 450. The bracketed numbers in the figure refer to the comments that follow.

Building Freestanding Applications under MVS

When building freestanding applications under MVS, CEE.V1R1M0.SEDCSPC must be included in the link-edit SYSLIB concatenation. Also, if C/370 library functions are needed, CEE.V1R1M0.SEDCSPC must precede CEE.V1R1M0.SCEELKED.

The routines to support this function (EDCXSTRT and EDCXSTRL) are CEESTART replacements in your module. Therefore, the appropriate EDCXSTR*n* routine must be explicitly included at the beginning of the module at link-edit.

A simple freestanding routine that requires a C/370 library function is shown in Figure 106 on page 448.

```
#include <stdio.h>
main() {
   puts("Hello, World");
   return 3999;
}
```

Figure 106. Sample Freestanding MVS Routine

This routine is compiled normally and link-edited using the control statements shown in Figure 107 on page 448. It is assumed that the object module is available to the link-edit using an OBJECT DD statement and that the intended member name is specified in the SYSLMOD DD statement. The CEE.V1R1M0.SCEERUN load library must be available at run-time because it contains the C/370 library function puts()

```
INCLUDE SYSLIB(EDCXSTRL)
INCLUDE OBJECT
ENTRY EDCXSTRL
```

Figure 107. Link-Edit Control Statements Used to Build a Freestanding MVS Routine

Figure 108 on page 449 shows how to compile and link a freestanding program using the cataloged procedure EDCCL.

Special Considerations for Reentrant Modules (VM)

A simple freestanding routine that does not require the LE/370 common library is shown in Figure 104 on page 447. This routine uses the exit() library function which, like sprintf(), is available to freestanding routines without requiring the LE/370 common library. This routine is not naturally reentrant, but the resulting load module will be reentrant.

```
#include <stdlib.h>

int main() {
    static int i[5]={0,1,2,3,4};
    exit(4320+i[1]);
}
```

Figure 104. Sample Reentrant Freestanding VM Routine

The commands required to build this routine are shown in Figure 105 on page 447. The bracketed numbers in the figure refer to the comments which follow.

```
CC RETS4321 (NOSTART RENT ...
GLOBAL TXTLIB SCEESPC SCEELKED CMSLIB          [105-1]
CPLINK EDCXSTRT RETS4321 EDCRCINT EDCXEXIT (MAP [105-2]
GLOBAL TXTLIB                                  [105-3]
LOAD CPOBJ (MAP RESET EDCXSTRT                 [105-4]
GENMOD RETS4321 (FROM EDCXSTRT
```

Figure 105. Building a Reentrant Freestanding VM Routine

Notes

[105-1] The TXTLIBs CMSLIB and SCEELKED are needed since CPLINK is a C/370 program. The SCEESPC TXTLIB is used to resolve external references.

[105-2] The alternate initialization routine (EDCXSTRT in this example) must be included explicitly in the module. This should be the first CSECT in the module.

The routine EDCRCINT must be explicitly included in the module as the RENT compile-time option is used. No error will be detected at load time if this routine is not explicitly included. At execution time, ABEND 2106, reason code 7205, will result if EDCRCINT is required but not included.

EDCXEXIT must be explicitly included if the exit() function is used in the application.

[105-3] No TXTLIB is required for further processing or execution of this module since no C/370 library functions are needed.

[105-4] EDCXSTRT must be specified as the module entry point.

EDCXSTRT as an alternate initialization routine:

```
LOAD EDCXSTRT main-function (RESET EDCXSTRT ...
GENMOD module-name (FROM EDCXSTRT
```

Figure 100. Specifying Alternate Initialization at Link-Edit

Another example of this is shown in Figure 101 on page 446.

Under MVS To explicitly include an alternative initialization routine under MVS, use the Linkage Editor INCLUDE and ENTRY control statements. To include the alternate initialization routines described in this chapter, CEE.V1R1M0.SCEESPC must be allocated to the SYSLIB DD. For example, the following linkage editor control stream might be used to specify EDCXSTRT as an alternate initialization routine:

```
INCLUDE SYSLIB(EDCXSTRT)
ENTRY EDCXSTRT
INCLUDE SYSIN
```

Figure 101. Specifying Alternate Initialization at Link-Edit Time

Another example of this is shown in Figure 107 on page 448.

Building Freestanding Applications under VM

When building freestanding applications under VM, SCEESPC TXTLIB must be made available (by the GLOBAL command) when issuing LOAD or INCLUDE commands. This TXTLIB is not required at execution time.

The routines to support this function (EDCXSTRT, and EDCXSTRL) are CEESTART replacements in your module. Therefore, the appropriate EDCXSTR*n* TEXT file must be explicitly included first in the module.

A simple freestanding routine that does not require the LE/370 common library is shown in Figure 102 on page 446. An example that requires the use of the C/370 Prelinkage Utility is shown in Figure 104 on page 447.

```
int main() {
   return 54321;
}
```

Figure 102. Simple Freestanding VM Routine

The VM commands required to build and run this routine are shown in Figure 103 on page 446.

```
CC RET54321 (NOSTART
GLOBAL TXTLIB SCEESPC
LOAD EDCXSTRT RET54321 (RESET EDCXSTRT
GENMOD RET54321 (FROM EDCXSTRT
RET54321
```

Figure 103. Building a Freestanding VM Routine

Appendix B. Building Applications in Systems Programming Environment

Note: The material in this section applies to C/370 applications only.

As a C/370 routine executes, facilities from the LE/370 common library are invoked to set up the execution environment in order to handle termination activities and provide storage management, error handling, run-time options parsing, ILC, and debugging support. In addition, the C/370 library functions are in the LE/370 common library.

For situations in which not all of these services are needed, the System Programming Facilities of C/370 can provide a limited environment.

System programming capabilities allow you to run applications without using the LE/370 common library, or with just the C/370 library functions, and to:

- develop applications in C that do not require the LE/370 common library on the machines on which the application runs.
- use C/370 as an Assembler language substitute to, for example, write exit routines for MVS, TSO, or JES.
- develop applications featuring:
  - a persistent C environment, in which a C/370 environment is created once and used repeatedly for C function execution from any language.
  - co-routines which use a two-stack model, as in client-server style applications. In this style, the user application calls on the applications server to perform services independently of the user and then return to the user.

For more information on the system programming facilities of C/370, see the *IBM SAA AD/Cycle C/370 Programming Guide.*

This chapter discusses how to build these applications once you have compiled them with the C/370 compiler. Note that you must compile these programs with the NOSTART option.

Building Freestanding Applications

Freestanding applications need to be linked with specific alternate initialization routines. This is accomplished differently depending on which operating system you compiled your application under.

Under VM: To explicitly include an alternative initialization routine under VM, include the TEXT file for the alternate entry point *first* in the LOAD commands. To include the alternate initialization routines described in this chapter, SCEESPC must be included in the GLOBAL TXTLIB list.

For example, the following sequence of commands might be used to specify

© Copyright IBM Corp. 1991

Usage Notes

1. If it is not possible to fit all of the information on one RENAME statement, one or more continuations may be used. For example, the L-name may be split across more than one statement. Continuations are enabled by placing a non-blank character in column 72 of the statement that is to be continued. They must begin in column 16 of the next statement.
2. A RENAME statement is ignored if the L-name is not encountered in the input.
3. A RENAME statement for an L-name is valid provided **all** of the following are true:
   - The L-name was not already mapped because of a rule that preceded the RENAME statement rule in the hierarchy described in "C/370 Prelinkage Utility Mapping of L-names to S-names" on page 433.
   - The L-name was not already mapped because of a previous valid RENAME statement for the L-name.
   - The S-name is not itself an L-name. This must be true even if the S-name has its own RENAME statement.
   - A previous valid RENAME statement did not rename another L-name to the same S-name.
   - Either the L-name or the S-name is not defined. Either the L-name and the S-name may be defined, but not both. This must be true even if the S-name has its own RENAME statement.

Note: The LIBRARY control statement is removed and not placed in the prelinkage output object module; the system linkage editor does not see the LIBRARY control statement.

RENAME Control Statement

Syntax

```
▸▸──RENAME──L-name──S-name──┬────────┬──────────────────▸◂
                            └─SEARCH─┘
```

where

L-name
: the name of the L-name that is to be renamed on output. All occurrences of this L-name are renamed.

S-name
: the name of the S-name to which the L-name will be changed. This name may be at most eight characters in length and case is respected.

SEARCH
: an optional parameter that requests that if the S-name is not defined, an attempt is made to search by automatic library call for the definition of the S-name.

The RENAME control statement is processed by the prelink utility and can be used for a number of purposes:

- to explicitly override the default name given to an L-name when an L-name is mapped to an S-name.

  You can explicitly control the names presented to the system linkage editor so that external variable and function names are consistent from one linkage editor run to the next. This consistency makes it easier to recognize control section and label names that may appear in system dumps and linkage editor listings. Another mapping rule (described in "C/370 Prelinkage Utility Mapping of L-names to S-names" on page 433) can provide the suitable name, but if linkage editor control section replacement is required, name consistency is required.

  Note that a RENAME control statement cannot be used to rename S-names.

- to explicitly bind an L-name to an S-name. This may sometimes be necessary, especially when communicating with objects from other language and assembler processors, since these processors generate only S-names.

RENAME control statements can be placed before, between, or after other control statements or object modules. It is acceptable to have an object module containing only RENAME statements. In addition, RENAME statements can be placed in input that is included as a result of other RENAME statements.

filename
: is the member of the DD to be included.

On MVS, the prelink utility processes INCLUDE statements in a manner similar to the MVS linkage editor with the following exceptions:

- INCLUDEs of identical member names are not allowed.
- INCLUDEs of both a DDNAME and a member from the same DDNAME are not allowed. The prelink utility ignores the second INCLUDE.

The VM LOAD command does not support INCLUDE statements. The prelink utility processes VM INCLUDE statements in the same manner as MVS statements with the following exceptions:

- Control statements of the form INCLUDE *name*, refer to a TEXT deck or TXTLIB member.
- Control statements of the form INCLUDE *name()*, refer to a DD called name.

LIBRARY Control Statement

Syntax

```
                          ┌──────────────┐
                          ↓              │
▸▸──LIBRARY──┬──name──(──┴──┬────────┬──┴──)──┬──────────────▸◂
             │              └─member─┘        │
             │    ┌──────────┐                │
             │    ↓          │                │
             └──(──external──┴──)─────────────┘
```

where

name
: Under VM, this is the name of a TXTLIB library. This could be a library updated by either the VM TXTLIB command or the C/370 C370LIB command.

  Under MVS, this is the name of a DD that defines a library. This could be a concatenation of one or more libraries created with or without the Object Library utility.

member
: the name of, or an alias of, a member of the specified library. Both S-names and L-names can be specified, so case distinction is significant.

  Under VM, automatic library calls search the given TXTLIB for the name instead of searching the TXTLIBs specified in the GLOBAL TXTLIB VM command.

  Under MVS, automatic library calls search the library and each subsequent library in the concatenation, if applicable and necessary, for the name instead of searching primary input.

external
: an external reference that may be unresolved after primary input processing. This external reference will not be resolved by Automatic Library call. Both S-names and L-names can be specified, so case distinction is significant.

- the ddname and, if applicable, the member name.

[4] `Static External Data Map`
: This section is generated if an object module was encountered that contains defined static external data (was compiled with the RENT compiler option). This section lists the names of such objects, their lengths, where they were mapped, and a FILE ID.

[5] `ESD Map of Defined and Long Names`
: This section lists the names of defined non-static external data objects and the list of L-names.

If the object is defined, a FILE ID is provided. Otherwise, this field is left blank. For any name, the input name and output S-name are listed. If the input name is indeed an L-name, the rule used to map the L-name to the S-name is applied. The rules are described in detail in "C/370 Prelinkage Utility Mapping of L-names to S-names" on page 433. If the name is not an L-name, this field is left blank.

Control Statement Processing

The only job control statements processed by the prelink utility are INCLUDEs, LIBRARYs, and RENAMEs. The remaining control statements are left unchanged until the link step.

Under VM, you can place your control statements in a file with a file type of TEXT and include this file as one of the input files to the prelinker, or you can have another INCLUDE control statement include the file. Under MVS, the control statements can be placed in the input stream or stored in a permanent data set.

INCLUDE Control Statement

Syntax

```
                  ┌─,──────────────────────────┐
                  ▼                            │
►►──INCLUDE──┬──filename──────────────────┬────┴──────────►◄
             │ ┌─,──────────────────────┐ │
             │ ▼                        │ │
             └─ddname──(──member──)─────┴─┘
```

where

filename
: is the name of the file to be included.

Under VM it looks for the file *filename* TEXT.

ddname
: is a ddname associated with a file to be included.

```
    30        4   00001   DFHCP011
    34        4   00001   DFHCP010
    38        4   00001   DFHBP025
    3C        4   00001   DFHBP024
    40        4   00001   DFHBP023
    44        4   00001   DFHBP022
    48        4   00001   DFHBP021
    4C        4   00001   DFHBP020
    50        4   00001   DFHEICB
    54        4   00001   DFHEID0
    58        4   00001   DFHLDVER
    60       28   00001   @STATIC
=====================================================================
|              ESD Map of Defined and Long Names                    | [5]
=====================================================================

                  OUTPUT
*REASON  FILE ID  ESD NAME   INPUT NAME

          00001   CEESTART   CEESTART
          00001   CEEMAIN    CEEMAIN
          00001   FUNCB      FUNCB
          00001   MAIN       MAIN

*REASON: P=#pragma or reserved    S=matches short name    R=RENAME card
         L=C Library              U=UPCASE option         D=Default


 ===========  E N D   O F   P R E - L I N K A G E   M A P  ============
```

[1] Heading

The heading is always generated and contains the product number, the compiler release number, the compiler version number, the date and the time the prelinkage step commenced, and a list of the prelinkage options in effect for the step.

On VM, the heading additionally shows the object module list specified on the command line. On MVS the SYSLIB ddname is assumed, so this is not shown. If the object module list is large, CPLINCL may be shown in place of the entire list.

[2] Object Resolution Warnings

This section is generated if objects were not undefined at the end of the prelinkage step, or if duplicate objects were detected during the step. The names of the applicable objects are listed.

[3] File Map *

This section lists only those object modules that were included in input. An object module consisting only of RENAME control statements, for example, is **not** shown. Also provided in this section are source origin (*ORIGIN), name (FILE NAME) and identifier (FILE ID) information. *ORIGIN indicates whether the object module came from primary input due to an INCLUDE control statement in primary or secondary input, due to a RENAME control statement or due to the resolution of L-name library references. The FILE ID may be found in other sections and is used as a cross reference to the object module. The FILE NAME may be:

- the data set name and, if applicable, the member name, or

```
1=================================================================
 |                    Pre-linkage Utility Map                    | [1]
 |                                                               |
 | CPLINK: 5688216 V1 R1 M00 IBM AD/Cycle C/370      91/10/17 17:57:14 |
 =================================================================

 Command Options. . . . . : NONCAL   NOMEMORY ER      DUP      MAP
                          : NOUPCASE


 =================================================================
 |                  Object Resolution Warnings                   | [2]
 =================================================================

 WARNING EDC4015: Unresolved references are detected:
 CEEBETBL EDCROOT  CEESG003


 =================================================================
 |                           File Map                            | [3]
 =================================================================

 *ORIGIN  FILE ID  FILE NAME

    P      00001   DD:SYSIN

 *ORIGIN:  P=primary input     PI=primary INCLUDE    SI=secondary INCLUDE
           A=automatic call     R=RENAME card         L=C Library


 =================================================================
 |                      Writable Static Map                      | [4]
 =================================================================

   OFFSET    LENGTH  FILE ID  INPUT NAME

        0         1   00001   DFHC0011
        4         1   00001   DFHC0010
        8         2   00001   DFHDUMMY
        C         2   00001   DFHB0025
       10         2   00001   DFHB0024
       14         2   00001   DFHB0023
       18         2   00001   DFHB0022
       1C         2   00001   DFHB0021
       20         2   00001   DFHB0020
       24         2   00001   DFHEIB0
       28         4   00001   DFHEIPTR
       2C         4   00001   STATINT
```

Table 57. Prelink options

Option	System Available: VM	System Available: MVS	Description
AUTO|NOAUTO	√		The AUTO option specifies that the prelink utility should search all virtual disks for TEXT files to resolve unresolved references. Use NOAUTO when using the CPOBJ file as input to the LINKLOAD EXEC.
DUP|NODUP	√	√	The DUP option detects duplicate symbol names. When duplicates are detected, the return code will minimally be set to a warning level of 4. NODUP does not affect the return code setting when duplicates are detected. Under VM, duplicate symbol names are directed to the terminal.
ER|NOER	√	√	The ER option detects external references. When external references are detected, the return code will minimally be set to a warning level of 4. NOER does not affect the return code setting when external references are detected. Under VM, external references are directed to the terminal.
LIBE|NOLIBE	√		The LIBE option specifies that the prelink utility should search TEXT libraries (specified previously with the VM GLOBAL command) to resolve unresolved references.
MAP|NOMAP	√	√	The MAP option specifies that the prelink utility should generate a prelink listing. See "C/370 Prelinkage Utility Listing" on page 438 for a description of the map.
MEMORY|NOMEMORY	√	√	The MEMORY option specifies that the prelink utility will buffer (retain in storage), for the duration of the prelink step, those object modules that are read and processed. The MEMORY option is used to increase prelink utility speed. To use this option, however, you may require additional memory. If you use this option and the prelink fails due to a storage error, you must increase your storage size or use the prelink utility without the MEMORY option.
NCAL|NONCAL		√	The NCAL option specifies that the prelink utility should not use automatic library call to resolve unresolved references. Automatic library call is performed when the NONCAL option is specified. Automatic library call applies to a library of user routines. The data set must be partitioned and must contain object modules. Automatic library call cannot apply to a library containing load modules.
UPCASE|NOUPCASE	√	√	The UPCASE option enforces the uppercase mapping of those L-names that are eight characters or less and have not been explicitly mapped by another mechanism. These L-names will be uppercased (with _ mapped to @) and names that begin with IBM or CEE will be changed to IB$ and CE$, respectively. The UPCASE option is useful when calling routines written in languages other then C. For example, COBOL and Assembler each uppercases all of its external names, so if the names are coded in lowercase in the C program and the LONGNAME option is used, the names will not match by default. The UPCASE option can be used to enforce this matching. The RENAME control statement can also be used for this purpose.

C/370 Prelinkage Utility Listing

The C/370 Prelinkage Utility produces a listing file called the Pre-linkage Utility Map when the MAP option is in effect. The default is to generate a listing file. The listing contains a number of individual sections, some of which are generated only if they are applicable.

In the sample that follows, the bracketed numbers refer to the comments that follow the map.

specify the following:

```
CPLINK filename POPT(MAP)...
```

When the prelink MAP option is specified (as opposed to the link option MAP), the prelink utility produces a file showing the mapping of static external data. This map shows name, length, and address information. If there are any unresolved references or duplicate symbols during the prelink step, they are displayed in the map.

PLIB
specifies the library names that are to be used by the automatic library call facility.

LOPT
is an output data set name.

Because the load module is generated by the linkage editor, you must use the linkage editor LOAD option in this parameter. If you do not specify an output data set name, a name is generated for you. The name generated by the CLIST consists of your user prefix followed by CPOBJ.LOAD(TEMPNAME).

The LOPT option allows you to pass linkage editor options to the linkage editor utility. For example, if you wanted the MAP option to be used by the prelink utility, and the NOMAP option to be used by the linkage editor, use the following CLIST command:

```
CPLINK filename POPT(MAP) LOPT(NOMAP...
```

LIB
is any additional libraries that you wish to use to resolve external references. These libraries are appended to the default C/370 library functions.

Examples: In the following example, assume your user prefix is RYAN, and the data set containing the input object module is a partitioned data set called RYAN.C.OBJ(INCCOMM). This example will generate a prelink listing without using the automatic call library. After the call, the load module is placed in a partitioned data set called RYAN.CPOBJ.LOAD(TEMPNAME) and the prelink listing is placed in a sequential data set called RYAN.CPOBJ.PMAP.

```
CPLINK OBJ('C.OBJ(INCCOMM)')
```

In the following example, assume that your user prefix is JERMAINE, and the data set containing the input object module is a partitioned data set called PHIL.C.OBJ(INCPYRL). This example will not generate a prelink listing, and the automatic call facility is to use the library HOGAN.LIB.SUB. The load module is placed in the partitioned data set JERMAINE.TBD.LOAD(MOD).

```
CPLINK OBJ('''PHIL.C.OBJ(INCPYRL)''')
       POPT(NOMAP,NCAL)
       PLIB('''HOGAN.LIB.SUB''')
       LOAD('TBD.LOAD(MOD)')
```

Prelink Options

The following table describes the prelink options available in C/370.

Invoking the Prelink Facility under TSO

The C/370 prelink utility is invoked under TSO through a CLIST. The IBM-supplied CLIST that invokes the prelink utility is called `CPLINK` and creates an executable module. If you want to create a reentrant load module, you must use the `CPLINK` CLIST instead of the `TSO LINK` command.

Syntax

```
▶▶──CPLINK──OBJ──(─┬───────────────────────┬─)──────────────▶
                   │      ┌──────,──────┐  │
                   │      ▼             │  │
                   └─'──────object──────┴─'┘

▶─┬─────────────────────────────────────────────────────┬──▶
  └─POPT──(─┬───────────────────────────┬─)─────────────┘
            │      ┌──option1──┐        │
            │      ├────,──────┤        │
            │      ▼           │        │
            └─'────option1─────┴─'──────┘

▶─┬─────────────────────────────────────────────────────┬──▶
  └─PLIB──(─┬───────────────────────────┬─)─────────────┘
            │      ┌──libname──┐        │
            │      ├────,──────┤        │
            │      ▼           │        │
            └─'────libname─────┴─'──────┘

▶─┬─────────────────────────────────────────────────────┬──▶
  └─LOPT──(─┬───────────────────────────┬─)─────────────┘
            │      ┌──option2──┐        │
            │      ├────,──────┤        │
            │      ▼           │        │
            └─'────option2─────┴─'──────┘

▶─┬─────────────────────────────────────────────────────┬──▶◀
  └─LIB──(─┬───────────────────────────┬─)──────────────┘
           │      ┌──libname──┐        │
           │      ├────,──────┤        │
           │      ▼           │        │
           └─'────libname─────┴─'──────┘
```

where

OBJ
: is an input data set name.

This is a required parameter. Each input data set must be a C object module compiled with the `RENT` or `LONGNAME` compile-time options, or a compiled program (C or otherwise) having no static external data.

POPT
: a string of prelink options.

The prelink utility options available for `CPLINK` are the same as for MVS batch. For example, if you want the `MAP` option to be used by the prelink utility,

Invoking the Prelink Facility under VM/CMS

Under VM, use the CPLINK EXEC to invoke the prelink utility.

Syntax

```
           ┌──────────┐
           ↓          │
▶▶──CPLINK────textdeck─┴──┬──────────────────┬──────▶◀
                          │    ┌────────┐    │
                          │    ↓        │    │
                          └─(────option─┴────┘
```

where

textdeck

is the file input to the prelink utility.

You can specify more than one file as input to the CPLINK EXEC. Each input file must be an object module with the file type TEXT (that is, a C program compiled with the RENT option or a compiled program, C or otherwise, having no static external data). If you previously used the VM GLOBAL TXTLIB command, you can specify the name of a TXTLIB member as the file name.

options

a list of options to be passed to the prelink facility.

The prelinker will be invoked with the CMOD EXEC if you specify options for the prelinker by using the option CPLINK(options) or if you have specified the LONGNAME compile-time option. See "Prelink Options" on page 437 for a list of pre-link options.

Examples: The following example prelinks the text decks ROUTER, SENDMSG and REPLYMSG and places the output text deck in CPOBJ TEXT A. A writable static map is generated and placed in CPOBJ RMAP A. Unresolved references are not processed.

```
CPLINK ROUTER SENDMSG REPLYMSG (NOLIBE
```

The following example prelinks the text decks SORT, MERGE and READFILE and displays only warning and error messages at the terminal. A prelink listing is not generated. All external references are resolved from ACCNT TXTLIB and a disk search.

```
GLOBAL TXTLIB ACCNT
CPLINK SORT MERGE READFILE (NOMAP NOER NODUP LIBE AUTO
```

To use CMOD to invoke the prelinker with the prelink options AUTO and the CMOD option AUTO, specify

```
CMOD PGM (CPLINK(AUTO) AUTO
```

Invoking the Prelink Facility under MVS Batch

The prelink utility is invoked by the EDCPL (prelink and link) cataloged procedure.

See Chapter 16, "Using IBM-supplied Cataloged Procedures" on page 62 for a description of IBM-supplied cataloged procedures.

even if a RENAME control statement for the name exists. For information on the RENAME control statement see "RENAME Control Statement" on page 443.

2. If the L-name was found to have a corresponding S-name, then the same name is chosen. For example, DOTOTALS is coded in both a C and assembler program. Under no circumstances is this name changed even if a RENAME statement for the name exists. This rule binds the L-name to its S-name.

3. If a valid RENAME statement for the L-name is present, then the S-name specified on the RENAME statement is chosen.

4. If the name corresponds to a C/370 library function or library object for which you did not supply a replacement, then the name chosen is the truncated and uppercased (with _ mapped to @) version of the L-name library name.

   In addition, this S-name is not chosen if either

   - A valid RENAME statement renames another L-name to this S-name. For example, the RENAME statement RENAME mybigname PRINTF would make the library printf() function unavailable if mybigname is found in input.
   - Another L-name is found to have the same name as this S-name. For example, explicitly coding and referencing SPRINTF in the C source program would make the library sprintf() function unavailable.

   Avoid such practices to ensure that the appropriate C/370 library function is chosen.

5. If the UPCASE option is specified, names that are eight characters or less are uppercased (with _ mapped to @). Names that begin with IBM or CEE will be changed to IB$, and CE$, respectively. Because of this rule, it is possible that two different names map to the same name. Care should therefore be exercised when using the UPCASE option. If a collision is found, a warning message is issued but the names are still mapped.

6. If none of the above rules apply, a default mapping is performed. This mapping is the same as the one the compile-time option NOLONGNAME uses for external names, taking collisions into account. That is, the name is truncated to eight characters and uppercased (with _ mapped to @). Names that begin with IBM or CEE will be changed to IB$ and CE$, respectively. If this name is the same as the original name then it is always chosen. This name is also chosen if a name collision does not occur. A name collision occurs if either

   - The S-name has already been seen in **any** input. That is, the name is not new.
   - After applying this default mapping, the same name is generated for at least two, previously unmapped names.

   If a collision occurs, a unique name is generated for the output name. For example, the name @ST00033 is manufactured.

An application that is compiled with the NOLONGNAME compile-time option and link-edited, with the exception of collisions, presents the linkage editor with the same names as when the application is compiled with the LONGNAME option and run.

- If the LIBE command option is in effect, the GLOBAL TXTLIBs are searched in order as follows:
  - If the TXTLIB contains a C370LIB-directory created using the C/370 Object Library Utility, and the C370LIB-directory indicates a defined symbol by that name exists, the member of the TXTLIB indicated as containing that symbol is read.
  - If the TXTLIB does not contain a C370LIB-directory created using the C/370 Object Library Utility and the reference is not to static external data, the member, or alias, with the same name as SNAME is read.

- The undefined name is an L-name
  - If the LIBE command option is in effect, the GLOBAL TXTLIBs are searched. If the TXTLIB contains a C370LIB-directory created using the C/370 Object Library Utility, and the C370LIB-directory indicates that a defined symbol by that name exists, the member of the TXTLIB indicated as containing that symbol is read.

On MVS, the following hierarchy is used to resolve a referenced and currently undefined symbol. In all cases, the symbol is only defined if it is indeed contained in the input from this process or in other future input:

- The undefined name is an S-name, for example SNAME.
  - If the NONCAL command option is in effect, the partitioned dataset(s) concatenated to SYSLIB are searched in order as follows:
    - If the data set contains a C370LIB-directory created using the C/370 Object Library Utility, and the C370LIB-directory indicates a defined symbol by that name exists, the member of the PDS containing that symbol is read.
    - If the data set does not contain a C370LIB-directory created using the C/370 Object Library Utility and the reference is not to static external data, the member, or alias, with the same name as SNAME is read.

- The undefined name is an L-name
  - If the NONCAL command option is in effect, the partitioned data set(s) concatenated to SYSLIB are searched. If the data set contains a C370LIB-directory created using the C/370 Object Library Utility, and the C370LIB-directory indicates a defined symbol by that name exists, the member of the PDS indicated as containing that symbol is read.

C/370 Prelinkage Utility Mapping of L-names to S-names

Because system linkage editors accept only S-names, the C/370 Prelinkage Utility maps L-names to S-names on output. S-names are not changed. L-names can be up to 255 characters in length; truncation of the L-names to the eight character S-name limit is therefore not sufficient because collisions may occur.

The C/370 Prelinkage Utility maps a given L-name to a S-name according to the following hierarchy:

1. If any occurrence of the L-name is a reserved run-time name, or was due to a #pragma map directive or #pragma csect directive, then that same name is chosen for all occurrences of the name. Under no circumstances is this name changed

- If the LIBE option is in effect, then SNAME is immediately resolved by reading the member of the first TXTLIB found in the GLOBALed list that has the same member name, or alias name.
- If SNAME is still unresolved, it may be subsequently resolved if a defined function or variable called SNAME is encountered in input.

- On MVS, primary input is resolved by reading the data set(s) allocated to the SYSIN DD.

INCLUDE control statements

- On VM,
  - For the INCLUDE *ddname()* and INCLUDE *ddname(member)* forms, an attempt is made to read the *ddname* or member of the *ddname* (whichever is specified). This request is resolved if the read is successful.
  - For the INCLUDE SNAME form, the input is resolved using the same algorithm as for primary input.

    See the appropriate link-edit and loader manual for a complete description of the INCLUDE control statement.
- On MVS, an attempt is made to read the DD or member of the DD (whichever is specified). This request is resolved if the read is successful.

References to currently undefined symbols (external references)

- If, during the automatic library call, the symbol was not found to be the name of an existing TXTLIB library routine or TEXT file, then the symbol can subsequently be defined if a function or variable with the same name is encountered.
- If the symbol is an L-name that was not resolved by automatic library call and for which a RENAME statement with the SEARCH option exists, the symbol is resolved under the S-name on the RENAME statement by automatic library call.

  See "RENAME Control Statement" on page 443 for a complete description of the RENAME control statement.
- On VM, if the symbol is an L-name that was not resolved by previous automatic library call and also corresponds to a C/370 Library function or object, the symbol is resolved under the S-name of the symbol. For example, if you do not supply a version of printf, an attempt would be made to find and use PRINTF in its place as the C/370 library only ships PRINTF.

Unresolved requests generate error or warning messages either to the terminal or to the prelinkage utility map.

C/370 Prelinkage Utility Automatic Call Library Processing

On VM, the following hierarchy is used to resolve a referenced and currently undefined symbol. In all cases, the symbol is only defined if it is contained in the input from this process or in other future input:

- The undefined name is an S-name, for example SNAME.
  - If the AUTO command option is in effect and the reference is not to static external data, SNAME TEXT is read.

Appendix A. Pre-linking Your C Application

The C/370 Prelinkage Utility combines the object modules that forms a C application and produces a single object module that can then be link-edited or loaded for execution. This utility must be used when one or both of the following are true:

- A C source file is compiled with the RENT compiler option.
- A C source file is compiled with the LONGNAME compiler option.

See the *IBM SAA AD/Cycle C/370 Programming Guide* for more information about these compile-time options.

For object modules from applications compiled with the RENT compiler option, the prelinkage utility:

- combines static initialization descriptors
- maps static storage
- removes static name and relocation information.

For object modules from applications compiled with the LONGNAME compiler option, the prelinkage utility maps L-names to S-names. L-names are mixed case external names of up to 255 characters in length while S-names are eight character, single-case external names.

C/370 Prelinkage Utility input can come from a number of sources; the utility generates a listing file.

Note: It is permissible to exclude object modules that do not refer to writable static or L-names during the prelink step. In fact, C/370 library functions are not included as part of automatic library calls. This omission of the C library functions can result in warning messages about unresolved references to C library functions or C library objects. Any unresolved C library functions or objects will be resolved in a subsequent link-edit step.

C/370 Prelinkage Utility Input Processing

The prelinkage utility includes input from a number of sources:

- primary input
- secondary input or object library search
- sources specified on one or more INCLUDE control statements encountered in primary and secondary input.

See the appropriate link-edit and loader manual for a complete description of INCLUDE.

The process of resolving or including input from these sources depends on the type of the source, and the current input and prelink options.

Primary input

- On VM, when an object module name SNAME is specified on the command line:
  - If it exists, it is immediately resolved by reading SNAME TEXT.

© Copyright IBM Corp. 1991

- A list of possible feedback codes that can be returned by the service to its caller
- Usage notes that provide additional needed information to the user
- A list of related callable services
- An example or examples of usage.

Chapter 41. Guidelines for Writing an LE/370 Callable Service

- Callable service parameters must follow the data type descriptions outlined in "Data Type Definitions" on page 262.
- Argument passing is by one level of indirection, either "by reference" or "by value". See "Methods for Passing Arguments to and from Routines" on page 10 for these argument passing styles.
- Avoid the use of operating system services and macros. Use LE/370 services whenever possible.
- Always use the prototype definition or the entry declaration whenever possible.
- Avoid using the CEE3SPM callable service ("CEE3SPM — Query and Modify LE/370 Hardware Condition Enablement" on page 318). CEE3SPM can change the condition handling semantics of the HLLs supported by LE/370.
- LE/370 assumes the following defaults for character strings:
  - For input arguments, a length-prefixed string (with the length of the 2-byte prefix not included in the length value)
  - For output arguments, a fixed-length string of 80 bytes, padded on the right with blanks as necessary.
- Allow a feedback code area to be optionally passed as the last parameter to the callable service. The feedback code must be a FEED_BACK data type and conform to the layout described in Chapter 18, "Communicating Conditions" on page 79.
- If omitted arguments are permitted by the HLL, a zero or NULL pointer must be used to indicate the omitted parameter in the parameter list that is passed to the callable service. For example:

address of parm1
address of parm2
0

  The last parameter passed in the list must have the high order bit on to indicate that it is last. If the last parameter is omitted, the zero value that the user passes in the parameter list must have the high order bit on, for example X'80000000'. Therefore, you must allow the user of the callable service to check for this bit when the last parameter passed to the service is omitted.
- When documenting callable services, follow the same general format used to document each of the callable services in this book. Each callable service description should contain (in this order):
  - A general description of what the service does
  - A diagram indicating the syntax of the call to the service
  - A complete description of each callable service parameter and an identification of the required data type

© Copyright IBM Corp. 1991

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

#define SUCCESS "\0\0\0\0"

int main (void) {

   _INT4 seed;
   _FLOAT8 random;
   _FEEDBACK fc;
   int number;

   seed = 0;
   CEERAN0(&seed,&random,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEERAN0 failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }
   number = random * 1000;
   printf("The 3 digit random number is %d\n",number);
}
```

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2ER	1	2523	The system time was not available when CEERAN0 was called. A seed value of 1 was used to generate a random number and a new seed value.
CEE2ES	3	2524	An invalid seed value was passed to CEERAN0. The random number was set to -1.

Usage Note

1. The uniform (0,1) pseudo-random numbers are generated as follows using the multiplicative congruential method:

```
seed(i) = (950706376 * seed(i-1)) mod 2147483647;

randomno(i) = seed(i) / 2147483647;
```

Examples

1. COBOL/370 Example —

```
77   SEED PIC S9(9) COMP.
77   RANDNUM COMP-2.
77   FC PIC X(12).
:

     MOVE 0 TO SEED.
     CALL "CEERAN0" USING SEED , RANDNUM , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>

int main(void) {

  _CHAR80 title = "This is the title of the dump report";
  _CHAR255 options = "THREAD(CURRENT) TRACEBACK FILES VARIABLES BLOCKS STORAGE";
  FILE *f;
  f = fopen("my.file","wb");
  fprintf(f,"my file record 1");

  CEE3DMP(title,options,NULL);
}
```

CEERAN0 — Calculate Uniform Random Numbers

The CEERAN0 service generates a sequence of uniform pseudo-random numbers between 0 and 1 using the multiplicative congruential method with a user-specified seed.

Syntax

```
▶▶──CEERAN0──(──seed──,──random_no──,──┬────┬──)──────────────▶◀
                                       └─fc─┘
```

seed (input/output)

a fullword signed integer representing an initial value used to generate random numbers.

seed must be a variable; it cannot be an input-only parameter. The valid range is 0 to +2,147,483,646.

If *seed*=0, the seed is generated from the current Greenwich Mean Time.

On return to the calling routine, CEERAN0 changes the value of *seed* so that it can be used as the new seed in the next call.

random_no(output)

an 8-byte double precision floating point number with a value between 0 and 1, exclusive.

If *seed* is invalid, *random-no* is set to –1 and CEERAN0 terminates with a non-zero feedback code.

fc(output)

an optional 12-byte feedback code.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE30U	2	3102	Unsupported or incompatible CEE3DMP options or suboptions were found and ignored.
CEE30V	3	3103	An error occurred in writing messages to the dump file.

The IBM-supplied default settings for the above options are

```
TRACEBACK THREAD(CURRENT) FILES VARIABLES NOBLOCKS NOSTORAGE
STACKFRAME(ALL) PAGESIZE(60) FNAME(CEEDUMP) CONDITION ENTRY
```

Usage Notes

1. Stackframe(n) includes intervening assembler or library routine stackframes, not just application routine stack frames. Therefore, setting a STACKFRAME value other than (ALL) may result in fewer application routines being dumped than expected.
2. CEE3DMP establishes a condition handler that captures all conditions which occur during dump processing. It terminates the section of the dump in progress when a condition occurs and inserts the following line in the dump:

   ```
   Exception occurred during dump processing at nnnnnnnn
   ```

   nnnnnnnn is the instruction address at the time of the exception. After this line is inserted in the report, dump processing continues with the next section of the dump.
3. COBOL and C file data is not produced or controlled by ILC.
4. CICS Considerations — The FNAME dump option is ignored under CICS. Dump output is written to a transient data queue named CESE.

Examples

1. COBOL/370 Example —

```
77   DMPTITL PIC X(80).
77   OPTIONS PIC X(255).
77   FC PIC X(12).
 :

     MOVE "This is the title of the dump report." TO DMPTITL.
     MOVE "NOTRACE NOVAR SF(0) NOCOND NOENTRY" TO OPTIONS.
     CALL "CEE3DMP" USING DMPTITL , OPTIONS , FC.
```

FNAME(ddname)
Specifies the ddname of the file to which the dump report is written.

The default ddname CEEDUMP is used if this option is not specified.

CONDITION
specifies that for each condition active on the call chain, the following information is dumped from the Condition Information Block (CIB):

- The address of the CIB
- The message associated with the current condition token
- The message associated with the original condition token, if different from the current one
- The location of the error
- The machine state at the time the condition manager was invoked
- The ABEND code and REASON code, if the condition occurred as a result of an ABEND.

The particular information that is dumped depends on the nature of the condition that caused the condition manager to be invoked. The machine state is included only if a hardware condition or ABEND occurred. The ABEND and REASON codes are included only if an ABEND occurred.

CONDITION may be abbreviated as COND.

NOCONDITION
Do not dump condition information for active conditions on the call chain.

NOCONDITION may be abbreviated as NOCOND.

ENTRY
Include in the dump a description of the routine which called CEE3DMP and the contents of the registers upon entry to CEE3DMP.

NOENTRY
Do not include in the dump a description of the routine that called CEE3DMP and the contents of the registers upon entry to CEE3DMP.

fc(output)
an optional 12-byte feedback code.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

VARIABLES may be abbreviated as VAR.

NOVARIABLES
Do not include a dump of variables, arguments, and registers.

NOVARIABLES may be abbreviated as NOVAR.

BLOCKS
Dump the control blocks used in LE/370 and member language libraries.

Global control blocks as well as control blocks associated with routines on the call chain are printed. Control blocks are printed for the routine that called CEE3DMP. The dump proceeds up the call chain for the number of routines specified by the STACKFRAME option (see below). Control blocks for files are also dumped if the FILES option was specified. See the FILES option above for more information.

BLOCKS may be abbreviated as BLOCK.

NOBLOCKS
Suppress the dump of control blocks.

NOBLOCKS may be abbreviated as NOBLOCK.

STORAGE
Dump the storage used by the program.

The storage is displayed in hexadecimal and character format. Global storage as well as storage associated with each routine on the call chain is printed. Storage is dumped for the routine that called CEE3DMP. CEE3DMP proceeds up the call chain for the number of routines specified by the STACKFRAME option. Storage for all file buffers is also dumped if the FILES option was specified (see above).

STORAGE may be abbreviated as STOR.

NOSTORAGE
Suppress storage dumps.

NOSTORAGE may be abbreviated as NOSTOR.

STACKFRAME(n|ALL)
Specifies the number of stack frames dumped from the call chain.

If STACKFRAME(ALL) is specified, all stack frames are dumped. No stack frame storage is dumped if STACKFRAME(0) is specified.

The particular information dumped for each stack frame depends on the VARIABLE, BLOCK, and STORAGE option declarations specified for CEE3DMP. The first stack frame dumped is the one associated with the routine which called CEE3DMP, followed by its caller, and proceeding backwards up the call chain.

STACKFRAME may be specified as SF.

PAGESIZE(n)
Specifies the number of lines on each page of the dump.

This value must be greater than 9. A value of 0 indicates that there should be no page breaks in the dump.

PAGESIZE may be abbreviated to PAGE. The default setting is PAGESIZE(60)

options
a 255-byte fixed-length character string enclosed in single quotes containing options describing the type, format, and destination of dump information.

Options are declared as a string of keywords separated by blanks or commas. Note that some options have suboptions that follow the option keyword and are contained in parentheses. The options may be specified in any order, but the last option declaration is honored if there is a conflict between it and any preceding options. The following options are recognized by LE/370:

THREAD(ALL|<u>CURRENT</u>)
Dump the current thread or all threads associated with the current enclave. The default setting of this option is to dump only the current thread. Only one thread is supported in Language Environment/370 Version 1.

THREAD may be abbreviated as THR. CURRENT may be abbreviated as CUR.

<u>TRACEBACK</u>
Include a traceback of all routines on the call chain. The traceback shows transfers of control from either calls or exceptions. The traceback extends backwards to the main program of the current thread.

TRACEBACK may be abbreviated as TRACE.

NOTRACEBACK
Do not include a traceback.

NOTRACEBACK may be abbreviated as NOTRACE.

<u>FILES</u>
Include attributes of all files that are open and the contents of the buffers used by the files. The particular attributes that are displayed are defined by the member languages.

FILES may be abbreviated as FILE.

NOFILES
Do not include file attributes of files that are open.

NOFILES may be abbreviated as NOFILE.

<u>VARIABLES</u>
Include a symbolic dump of all variables, arguments, and registers.

Variables include arrays and structures. Register values are those saved in the stack frame at the time of call. There is no way to print a subset of this information.

Variables and arguments are printed only if the symbol tables are available. A symbol table is generated if a program is compiled using the compile options shown below for each HLL:

Language	***Compile Option***
C	TEST(SYM)
COBOL	TEST or TEST(h,SYM)

The variables, arguments, and registers are dumped starting with the routine that called CEE3DMP. The dump proceeds up the chain for the number of routines specified by the STACKFRAME option. See below for a description of the STACKFRAME option.

Example

1. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

#define SUCCESS "\0\0\0\0"

int main (void) {

   int x,y,z;
   _VSTRING commands;
   _FEEDBACK fc;

   strcpy(commands.string,"AT LINE 30 { LIST(x); LIST(y); } GO;");
   commands.length = strlen(commands.string);

   CEETEST(&commands,&fc);

   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEETEST failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }
   x = y = 12;
   .
   .
   .
   .
   .
   .
   /* debug tool will display the values of x and y at statement 30 */
   .
   .
   .
}
```

CEE3DMP — Generate Dump

The CEE3DMP callable service generates a dump of the run-time environment of LE/370 and the member language libraries. Sections of the dump are selectively included, depending on options specified with the *options* parameter. Output from CEE3DMP is written to the default ddname CEEDUMP, unless you specify the ddname of another file using the FNAME option of CEE3DMP.

Syntax

```
▸▸──CEE3DMP──(──title──,──options──,──┬────┬──)──────────────▸◂
                                      └─fc─┘
```

title (input)

an 80-byte fixed-length character string containing a title to be printed at the top of each page of the dump.

tool command list, refer to *AD/Cycle CODE/370 User's Guide*. If you want to invoke another interactive debug service, refer to the appropriate User's Guides.

If this parameter is omitted, your debug service defines the action taken. For more information, refer to the appropriate User's Guides for your debug service.

fc(output)
an optional 12-byte feedback code.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2F2	3	2530	A debug tool was not available.

2. C/370 Example —

```
#include <leawi.h>

#define SUCCESS "\0\0\0\0"
typedef struct {
   int value1,value2,value3;
   char slot1Y80";
} info_struct;

int main (void) {

   _INT4 function_code, field_number, field_value;
   _FEEDBACK fc;
   struct info_struct *info;
   info = malloc(sizeof(info_struct));
   .
   .
   .
   /* set user field 1 to be a pointer to an info_struct */
   function_code = 1;
   field_number = 1;
   field_value = (int)info;

   CEE3USR(&function_code,&field_number,&field_value,&fc);

   .
   .
   .
   /* get the value of field 2 */
   function_code = 2;
   field_number = 1;

   CEE3USR(&function_code,&field_number,&field_value,&fc);
   .
   .
   .
}
```

CEETEST — Invoke Debug Tool

The CEETEST callable service invokes a debug service such as AD/Cycle CODE/370.

Syntax

```
▸▸──CEETEST──(──┬────────────────────┬──,──┬────┬──)──────────▸◂
                └─string_of_commands─┘     └─fc─┘
```

string_of_commands (input)

a halfword prefixed string containing a debug tool command list. *string_of_commands* is optional.

If a debug tool is available, the commands in the list are passed to the debug tool and carried out. If you are using CEETEST to invoke AD/Cycle CODE/370 and need more information about how to create an AD/Cycle CODE/370 debug

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3PS	3	3900	The function code passed to CEE3USR was not 1 or 2.
CEE3PT	3	3901	The field number passed to CEE3USR was not 1 or 2.

Usage Note

1. LE/370 initializes both user area fields to X'00000000' during enclave initialization.

Examples

1. COBOL/370 Example —

```
77   FUNCODE PIC S9(9
77   FIELDNO PIC S9(9
77   INVALUE PIC S9(9
77   FC PIC X(12).
 :

     MOVE 2 TO FUNCODI
     MOVE 1 TO FIELDN(
     CALL "CEE3USR" U!                         JE , FC.
```

[illegible] 3/4/92

Can also get

CEE3I1

3649

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>

int main(int argc, char * argv[]") {

   _CHAR80 parm;

   CEE3PRM(parm,NULL);
   printf("%s\n",parm);
}
```

CEE3USR — Set or Query User Area Fields

The CEE3USR callable service sets or queries one of two 4-byte fields known as the *user area* fields. The user area fields are associated with an enclave and are maintained on an enclave basis. A user area might be used by vendor or application programs to store a pointer to a global data area or keep a recursion counter.

Syntax

```
►►──CEE3USR──(──function_code──,──field_number──,──field_value──,──►

►──┬────┬──)──────────────────────────────────────────────────────►◄
   └─fc─┘
```

function_code(input)
a fullword binary integer representing the function to be performed:

1 = SET user area field according to the value specified in *field_value*
2 = QUERY user area field; return current value in *field_value*

field_number(input)
A fullword binary integer indicating the field to set or query. *field_number* must be specified as either 1 or 2.

field_value(input/output)
a fullword binary integer.

If *function_code* is specified as 1 (meaning SET user area field), *field_value* contains the value to be copied to the user area.

If *function_code* is specified as 2 (meaning query user field area), the value in the user area is copied to *field_value*.

fc(output)
an optional 12-byte feedback code.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine.

Chapter 40. General LE/370 Callable Services

CEE3PRM — Query Parameter String

The CEE3PRM callable service returns to the calling routine the parameter string that was specified at invocation of the program. The returned parameter string contains only program arguments. If no program arguments are available, a blank string is returned.

Syntax

```
▸▸——CEE3PRM—(—char_parm_string—,—┬────┬—)——————————————▸◂
                                 └─fc─┘
```

char_parm_string (output)

an 80-byte fixed length string that is passed to CEE3PRM.

The *char_parm_string* that is passed to this service contains the parameter string that was specified at invocation of the program. If this parameter string is longer than 80 characters, it is truncated. If the parameter string is shorter than 80 characters, the returned string is padded with blanks. If the program argument passed to the service is absent, or is not a character string, *char_parm_string* is blank.

fc (output)

an optional 12-byte feedback code.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, a condition can be signaled. The possible conditions are:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Usage Note

1. C/370 Consideration — C users can use the __sysplist macro to return a program argument that is longer than 80 characters. See Chapter 3, "Parameter List Formats" on page 10 for more information on program arguments.

Examples

1. COBOL/370 Example —

```
77   PARMSTR PIC X(80).
77   FC PIC X(12).
:

     CALL "CEE3PRM" USING PARMSTR , FC.
```


© Copyright IBM Corp. 1991

C/370 Example

```
/*****************************************************************/
/* The example shows calls to the LE/370 math routines CEESDLG1 and   */
/* CEESIMOD.  They are typical of how all of the LE/370 Mathematical  */
/* Routines are used.                                                 */
/*****************************************************************/
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main (void) {

  _FLOAT8 f1,result;
  _INT4 int1, int2, intr;

  _FEEDBACK fc;
  #define SUCCESS "\0\0\0\0"

  f1 = 1000.0;

  CEESDLG1(&f1,&fc,&result);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEESDLG1 failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }

  printf("%f log base 10 is %f\n",f1,result);

  int1 = 39;
  int2 = 7;
  CEESIMOD(&int1,&int2,&fc,&intr);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEESIMOD failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }

  printf("%d modulo %d is %d\n",int1,int2,intr);
}
```

A Call to the CEESIXPI routine:

```
77   ARG1IS PIC S9(9) COMP.
77   ARG2IS PIC S9(9) COMP.
77   ARG1RS COMP-1.
77   ARG2RS COMP-1.
77   ARG1RL COMP-2.
77   ARG2RL COMP-2.
77   RESLTIS PIC S9(9) COMP.
77   RESLTRS COMP-1.
77   RESLTRL COMP-2.


     CALL "CEESIXPI" USING ARG1IS , ARG2IS , FBCODE , RESLTIS.
```

A Call to the CEESSXPI routine:

```
77   ARG1IS PIC S9(9) COMP.
77   ARG2IS PIC S9(9) COMP.
77   ARG1RS COMP-1.
77   ARG2RS COMP-1.
77   ARG1RL COMP-2.
77   ARG2RL COMP-2.
77   RESLTIS PIC S9(9) COMP.
77   RESLTRS COMP-1.
77   RESLTRL COMP-2.


     CALL "CEESSXPI" USING ARG1RS , ARG2IS , FBCODE , RESLTRS.
```

A Call to the CEESSAT2 routine:

```
77   ARG1IS PIC S9(9) COMP.
77   ARG2IS PIC S9(9) COMP.
77   ARG1RS COMP-1.
77   ARG2RS COMP-1.
77   ARG1RL COMP-2.
77   ARG2RL COMP-2.
77   RESLTIS PIC S9(9) COMP.
77   RESLTRS COMP-1.
77   RESLTRL COMP-2.

     CALL "CEESSAT2" USING ARG1RS , ARG2RS , FBCODE , RESLTRS.
```

COBOL Considerations

Some of the COBOL/370 intrinsic functions perform the same functions as the LE/370 math routines, and produce the same results, although the interface requirements of the COBOL/370 intrinsic functions and the LE/370 math routines differ.

For a list and detailed description of COBOL/370 intrinsic functions that produce the same results as LE/370 math routines, see the *IBM SAA AD/Cycle COBOL/370 Programming Guide.*

Examples

COBOL/370 Examples

A Call to the CEESSLOG routine:

```
77    ARG1IS PIC S9(9) COMP.
77    ARG2IS PIC S9(9) COMP.
77    ARG1RS COMP-1.
77    ARG2RS COMP-1.
77    ARG1RL COMP-2.
77    ARG2RL COMP-2.
77    RESLTIS PIC S9(9) COMP.
77    RESLTRS COMP-1.
77    RESLTRL COMP-2.

    CALL "CEESSLOG" USING ARG1RS , FBCODE , RESLTRS.
```

A Call to the CEESDLG1 routine:

```
77    ARG1IS PIC S9(9) COMP.
77    ARG2IS PIC S9(9) COMP.
77    ARG1RS COMP-1.
77    ARG2RS COMP-1.
77    ARG1RL COMP-2.
77    ARG2RL COMP-2.
77    RESLTIS PIC S9(9) COMP.
77    RESLTRS COMP-1.
77    RESLTRL COMP-2.

    CALL "CEESDLG1" USING ARG1RL , FBCODE , RESLTRL.
```

Table 56. Math Routine Feedback Codes

Feed-back Code	Severity	Error code	Message text
CEE1UI	2	2002	The argument value was too close to one of the singularities (plus or minus pi/2, plus or minus 3pi/2, for the tangent; or plus or minus pi, plus or minus 2pi, for the cotangent) in math routine *routine-name*.
CEE1UJ	2	2003	For an exponentiation operation (I**J) where I and J are integers, I was equal to zero and J was less than or equal to zero in math routine *routine-name*.
CEE1UK	2	2004	For an exponentiation operation (R**I) where R is real and I is integer, R was equal to zero and I was less than or equal to zero in math routine *routine-name*.
CEE1UL	2	2005	The value of the argument was outside the valid range *range* in math routine *routine-name*.
CEE1UM	2	2006	For an exponentiation operation (R**S) where R and S are real values, R was equal to zero and S was less than or equal to zero in math routine *routine-name*.
CEE1UN	2	2007	The exponent exceeded *limit* in math routine *routine-name*.
CEE1UQ	2	2010	The argument was less than *limit* in math routine *routine-name*.
CEE1UR	2	2011	The argument was greater than *limit* in math routine *routine-name*.
CEE1US	2	2012	The argument was less than or equal to *limit* in math routine *routine-name*.
CEE1UU	2	2014	Both arguments were equal to *limit* in math routine *routine-name*.
CEE1V0	2	2016	The absolute value of the argument was greater than *limit* in math routine *routine-name*.
CEE1V1	2	2017	The absolute value of the argument was greater than or equal to *limit* in math routine *routine-name*.
CEE1V4	2	2020	For an exponentiation operation (R**S) where R and S are real values, R was less than zero and S was not an integer or the absolute value of S was an integer greater than *limit* in math routine *routine-name*.
CEE1V5	2	2021	For an exponentiation operation (X**Y), the argument combination of Y*log2(X) generated a number greater than or equal to *limit* in math routine *routine-name*.
CEE1V9	1	2025	An underflow has occurred in in math routine *routine-name*.

Note: The values assigned to *limit* and *routine-name* vary depending on the routine that was called. The value assigned to *limit* will be the value indicated as the output range in Table 55 on page 409.

Feedback Code Descriptions

Feedback codes are used in writing condition handlers for your application. Feedback codes are returned from math routines, in the form of condition tokens, when you specify a feedback parameter in the routine. This section helps you determine feedback codes for the conditions you want to handle.

To identify a feedback code, first find the error code for the associated math routine in Table 55 on page 409 . Then use the following table to identify the feedback code.

Table 55 (Page 1 of 2). Math Routine Descriptions

Definition	Routine Name	Parameter Type	Equation	No. Inputs	Input Range	Output Range[1]	Error Code
Error function	CEESxERF	S D	$y=\frac{2}{\sqrt{\pi}}\int_0^x e^{-u^2}du$	1	any value	$-1 \le y \le 1$	None
Gamma and log gamma	CEESxGMA	S D	$y=\int_0^\infty u^{x-1}e^{-u}du$	1	$x > 2^{-252}$ and $x < 57.5744$	$0.88560 \le y \le \Omega$	2005
	CEESxLGM	S D	$y=\log_e \Gamma(x)$ or $y=\log_e \int_0^\infty u^{x-1}e^{-u}du$	1	$x > 0$ and $x < 4.2913 \bullet 10^{73}$	$-0.12149 \le y \le \Omega$	2005
Square root	CEESxSQT	S D	$y=\sqrt{x}$ or $y=x^{1/2}$	1	$x \ge 0$	$0 \le y \le \Omega^{1/2}$	2010
Modular arithmetic	CEESxMOD	I S D	$y=x_1$ (modulo x_2) Note 4	2	$x_2 \ne 0$ Note 5		None
Truncation	CEESxINT	S D	$y=$ (sign of x) $\bullet\ n$ where $n=[\,\lvert x \rvert\,]$ Note 6	1	any value		None
Nearest whole number	CEESxNWN	S D	$y=$ (sign of x) $\bullet\ v$ where $v=[\,\lvert x+.5 \rvert\,]$ if $x \ge 0$ or $v=[\,\lvert x-.5 \rvert\,]$ if $x < 0$. Note 6	1	any value		None
Nearest integer	CEESxNIN	S[2] D[2]	$y=$ (sign of x) $\bullet\ n$ where $n=[\,\lvert x+.5 \rvert\,]$ if $x \ge 0$ or $n=[\,\lvert x-.5 \rvert\,]$ if $x < 0$. Note 7	1	any value		None
Positive difference	CEESxDIM	I S D	$y=x_1-x_2$ if $x_1 > x_2$ $y=0$ if $x_1 \le x_2$	2	any value		None
Transfer of sign	CEESxSGN	I S D	$y=\lvert x_1 \rvert$ if $x_2 \ge 0$ $y=-\lvert x_1 \rvert$ if $x_2 < 0$	2	any value		None

Notes:

[1] $\Omega = 16^{63}(1-16^{-6})$ for single precision routines, and $16^{63}(1-16^{-14})$ for double precision routines.

[2] The output parameter is type I.

[3] The argument for the cotangent routines may not approach a multiple of π; the argument for the tangent routines may not approach an odd multiple of $\pi/2$.

[4] The expression x_1(modulo x_2) is defined as $x_1-[x_1/x_2] \bullet x_2$, where the brackets indicate that an integer is used. The largest integer whose magnitude does not exceed the magnitude of x_1/x_2 is used. The sign of the integer is the same as the sign of x_1x_2.

[5] If $x_2=0$, the modulus routine is undefined. In addition, a divide exception is recognized and an interruption occurs.

[6] $[\,\lvert x \rvert\,]$ is such that $v=\lvert m \rvert$, where m is the greatest integer satisfying the relationship $\lvert m \rvert \le \lvert x \rvert$, and the resulting v is expressed as a real value.

[7] $[\,\lvert x \rvert\,]$ is such that $n=\lvert m \rvert$, where m is the greatest integer satisfying the relationship $\lvert m \rvert \le \lvert x \rvert$.

Table 55 (Page 1 of 2). Math Routine Descriptions							
Definition	**Routine Name**	**Parameter Type**	**Equation**	**No. Inputs**	**Input Range**	**Output Range**[1]	**Error Code**
Common and natural logarithm	CEESxLOG	S D	$y = \log_e x$ or $y = \ln x$	1	$x > 0$	$y \geq -180.218$ $y \leq 174.673$	2012
	CEESxLG1	S D	$y = \log_{10} x$	1	$x > 0$	$y \geq -78.268$ $y \leq 75.859$	2012
	CEESxLG2	S D	$y = \log_2 x$	1	$x > 0$	$y \geq -260$ $y < 252$	2012
Exponential	CEESxEXP	S D	$y = e^x$	1	$x \leq 174.673$	$0 \leq y \leq \Omega$	2011
	CEESxXPI	I S D	$z = x^y$	2	if $x = 0$ then $y > 0$ or if $x < 0$ then $y =$ an integer	$-\Omega \leq z \leq \Omega$	2003, 2004, 2005, 2006, 2007, 2020, 2021
	CEESxXPS	S	$z = x^y$	2	if $x = 0$ then $y > 0$ or if $x < 0$ then $y =$ an integer	$-\Omega \leq z \leq \Omega$	2003, 2004, 2005, 2006, 2007, 2020, 2021
	CEESxXPD	D	$z = x^y$	2	if $x = 0$ then $y > 0$ or if $x < 0$ then $y =$ an integer	$-\Omega \leq z \leq \Omega$	2003, 2004, 2005, 2006, 2007, 2020, 2021
Sine and cosine	CEESxSIN	S D	$y = \sin(x)$	1	$\mid x \mid < (2^{18} \bullet \pi)$ (in radians)	$-1 \leq y \leq 1$	2017
	CEESxCOS	S D	$y = \cos(x)$	1	$\mid x \mid < (2^{18} \bullet \pi)$ (in radians)	$-1 \leq y \leq 1$	2017
Tangent and cotangent	CEESxTAN	S D	$y = \tan(x)$	1	$\mid x \mid < (2^{18} \bullet \pi)$ (in radians) Note 3	$-\Omega \leq y \leq \Omega$	2002, 2017
	CEESxCTN	S D	$y = \cot(x)$	1	$\mid x \mid < (2^{18} \bullet \pi)$ (in radians) Note 3	$-\Omega \leq y \leq \Omega$	2002, 2017
Arcsine and arccosine	CEESxASN	S D	$y = \arcsin(x)$	1	$\mid x \mid \leq 1$	$-\pi/2 \leq y \leq \pi/2$ (in radians)	2016
	CEESxACS	S D	$y = \arccos(x)$	1	$\mid x \mid \leq 1$	$0 \leq y \leq \pi$ (in radians)	2016
Arctangent	CEESxATN	S D	$y = \arctan(x)$	1	any value	$-\pi/2 \leq y \leq \pi/2$ (in radians)	2017
	CEESxAT2	S D	$y = \arctan\left(\frac{x_1}{x_2}\right)$	2	$x_2 > 0$	$-\pi < y \leq \pi$	2014
Hyperbolic sine and cosine	CEESxSNH	S D	$y = \sinh(x)$	1	$\mid x \mid < 175.366$	$-\Omega \leq y \leq \Omega$	2017
	CEESxCSH	S D	$y = \cosh(x)$	1	$\mid x \mid < 175.366$	$-\Omega \leq y \leq \Omega$	2017
Hyperbolic tangent	CEESxTNH	S D	$y = \tanh(x)$	1	any value	$-1 \leq y \leq 1$	None
	CEESxATH	S D	$y = \tanh^{-1}(x)$	1	$\mid x \mid < 1$	$-\Omega \leq z \leq \Omega$	2017
Absolute value	CEESxABS	I S D	$y = \mid x \mid$	1	any value	$0 \leq y \leq \Omega$	None

Error Code: This column gives the number of the message issued when an error occurs. See Table 56 on page 412 for the Feedback Code and message text associated with a given message number. The error messages are described further in *Language Environment/370 Debugging and Run-time Messages Guide*.

Note: In the table, the following approximate values are represented by

$(2^{18} \cdot \pi)$ and $(2^{50} \cdot \pi)$:

$(2^{18} \cdot \pi) = .82354966458264280 + 06$
$(2^{50} \cdot \pi) = .35371188760142200 + 16$

Table 55 on page 409 contains information on what the declared type of this parameter must be.

For example, the statement

```
CALL CEESSLOG (X,,Y)
```

passes the value of the 32-bit float variable X to CEESSLOG. CEESSLOG which computes the natural logarithm (base e) of X, and stores the result in the 32-bit float variable Y. Since the feedback code parameter is not specified, no feedback code is returned.

Math Routine Descriptions

In the table that follows, the following information is provided for each math routine:

Definition: This column states the nature of the computation performed by the routine.

Routine Name: This column gives the routine names of the routine.

Parameter Type: This column describes the acceptable parameter types that are input to and output by the specified routine, and is indicated by the following characters:

I 32-bit binary integer

S 32-bit single float number

D 64-bit double float number

Note: The characters correspond to the fifth character of the called routine.

The output parameter type is the same as the input parameter types, except where noted by a footnote in Table 55 on page 409. For routines that have two input parameters, the two parameters must be of the same type (except where noted).

Equation: This column gives a math equation that represents the computation. An alternative equation is given in those cases in which there is another way of representing the computation in math notation. For example, the square root can be represented either as:

$y=\sqrt{x}$ or $y=x^{1/2}$

No. Inputs: This column states how many input parameters must be passed to the routine. See page 406 for the appropriate syntax.

Input Range: This column gives the valid range for input parameters. If a parameter is not within the range, an error message is issued. (See Error Code in Table 55 on page 409.)

Output Range: This column gives the valid range for the output parameter returned by the routine. Type notation used is the same as that for the argument type.

Chapter 39. Math Routines

The LE/370 math routines provide standard math computations. They can be called either from LE/370-conforming languages or, if the common run-time environment is initialized, from assembler routines.

Calling the Math Routines

LE/370 routines are invoked by using the Cross System Consistent call interface, or through the conventional call statement specific to the S/370 high level language being used. With either of the call interfaces, you may use one of two syntax descriptions for the math routines, depending on how many input parameters are required by the routine. The syntax descriptions are the following:

Syntax - 1 Input Parameter

```
▶▶──CEESxnnn──(──parm1──,──┬────┬──,──result──)──────────▶◀
                           └─fc─┘
```

Syntax - 2 Input Parameters

```
▶▶──CEESxnnn──(──parm1──,──parm2──,──┬────┬──,──result──)──────▶◀
                                     └─fc─┘
```

CEES*xnnn*
The name of the math routine being called. The character *x* identifies the type of parameters being passed to the routine, and may be one of the following:

I 32-bit binary integer

S 32-bit single float number

D 64-bit double float number

The character *nnn* identifies the routine being called. Table 55 on page 409 contains information on the characters for the various routines.

parm1
The first input parameter to the routine. The declared type of this parameter must match the type specified by the *x* in the called routine.

parm2
The second input parameter to the routine. Except where noted, the declared type of this parameter must match the type specified by the *x* in the called routine.

fc If this optional parameter is specified, a 12-byte feedback code is returned from the routine in the form of a condition token. Otherwise, the condition token is passed to the condition handler if a condition occurs.

LE/370 condition handling actions are described in detail in Chapter 19, "Condition Management" on page 83.

result
The output parameter containing the result of the computations performed by the routine.

© Copyright IBM Corp. 1991

```
MOVE "US" TO COUNTRY.
CALL "CEEFMDA" USING COUNTRY , INPIC-STRING , FMDAFC.
DISPLAY "returned date picture string :"  , INPIC-STRING.

MOVE 152385 TO LILDATE.
MOVE 80 TO INPIC-LENGTH.
CALL "CEEDATE" USING LILDATE , INPIC , DATEPIC , DATEFC.
DISPLAY "Jan 1 2000 returned in US date format :" , DATEPIC.
```

Figure 96. COBOL/370 example illustrating calls to CEEFMDA and CEEDATE

```
memcpy ( COUNTRY , "US" , 2 );
CEEFMDA ( COUNTRY , INPIC.string , &FMDAFC );
printf ( "returned date picture string : %.50s\n" , INPIC.string );

LILDATE = 152385;
INPIC.length = 80;
CEEDATE ( &LILDATE , &INPIC , DATEPIC , &DATEFC );
printf ( "Jan 1 2000 returned in US date format : %.50s\n" , DATEPIC );
```

Figure 97. C/370 example illustrating calls to CEEFMDA and CEEDATE

Figure 98 and Figure 99 on page 405 contain an example using the CEEFMDT and CEEDATM callable services to obtain the default date and time format for the United States, then use this to convert the number of seconds since 00:00:00 14 October 1582 to a date and time expressed in the United States format.

```
MOVE "US" TO COUNTRY.
CALL "CEEFMDT" USING COUNTRY , PICSTR-STRING , FMDTFC.
DISPLAY "returned date/time picture string :"  ,
PICSTR-STRING.
MOVE 13166064000.0 TO SECS.
MOVE 80 TO PICSTR-LENGTH.
CALL "CEEDATM" USING SECS , PICSTR , DTSTAMP , DATMFC.
DISPLAY "Jan 1 2000 midnight returned in US time format :"  ,
DTSTAMP.
```

Figure 98. COBOL/370 example illustrating calls to CEEFMDT and CEEDATM

```
memcpy ( COUNTRY , "US" , 2 );
CEEFMDT ( COUNTRY , PICSTR.string , &FMDTFC );

printf ( "returned date/time picture string : %.80s\n" , PICSTR.string );

SECS = 13166064000.0;
PICSTR.length = 80;
CEEDATM ( &SECS , &PICSTR , DTSTAMP , &DATMFC );

printf ( "Jan 1 2000 midnight returned in US time format : %.80s\n" ,
DTSTAMP );
```

Figure 99. C/370 example illustrating calls to CEEFMDT and CEEDATM

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>
#define SUCCESS "\0\0\0\0"

void payend (_VSTRING *date, _VSTRING *pic, _VSTRING *paydate) {

    _FEEDBACK fc;
    _INT4 day, lilian;
    _CHAR80 tempdate;
    int adjust[7]" = {5,4,3,2,1,0,6};

    /* convert the input date to Lilian format */
    CEEDAYS(date,pic,&lilian,&fc);

    if (memcmp(&fc,SUCCESS,4) != 0) {
       printf("CEEDAYS failed with message %d\n",fc.tok_msgno);
       paydate = NULL;
       return;
    }

    /* get the day of the week */
    CEEDYWK(&lilian,&day,&fc);


    if (memcmp(&fc,SUCCESS,4) != 0) {
       printf("CEEDYWK failed with message %d\n",fc.tok_msgno);
       paydate = NULL;
       return;
    }

    /* adjust to the next Friday */

    day = lilian + adjust[day-1];      /* array is 0 based */

    /* convert back the input format */
    CEEDATE(&day,pic,tempdate,&fc);

    if (memcmp(&fc,SUCCESS,4) != 0) {
       printf("CEEDATE failed with message %d\n",fc.tok_msgno);
       paydate = NULL;
       return;
    }

    memcpy(paydate->string,tempdate,pic->length);
    paydate->length = pic->length;
    return;
}
```

Figure 95. C/370 example illustrating calls to CEEDAYS, CEEDYWK, and CEEDATE

Figure 96 on page 404 and Figure 97 on page 404 contain an example using the CEEFMDA and CEEDATE callable services to obtain the default date format for the United States, then use this to convert a Lilian date to a date expressed in the United States format.

```
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEISEC failed with message %d\n",fc.tok_msgno);
      newdate = NULL;
      return;
   }

   /* convert the lilian date the input format */
   CEEDATM(&sec1,pic,tempdate,&fc);

   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEDATM failed with message %d\n",fc.tok_msgno);
      newdate = NULL;
      return;
   }
   /* set up the return string */
   memcpy(newdate->string,tempdate,pic->length);
   newdate->length = pic->length;
   return;
}
```

Figure 93 (Part 2 of 2). C/370 example illustrating calls to CEESECS, CEESECI, CEEISEC, and CEEDATM

Figure 94 and Figure 95 on page 403 contain COBOL and C examples, respectively, which use CEEDAYS, CEEDYWK, and CEEDATE to compute the pay ending date (the following Friday) for any given date.

```
           MOVE 0 TO ADJUST (1).
           MOVE 5 TO ADJUST (2).
           MOVE 4 TO ADJUST (3).
           MOVE 3 TO ADJUST (4).
           MOVE 2 TO ADJUST (5).
           MOVE 1 TO ADJUST (6).
           MOVE 0 TO ADJUST (7).
           MOVE 6 TO ADJUST (8).

           MOVE "YYMMDD" TO PICSTR-STRING.
           MOVE 80 TO PICSTR-LENGTH.
           CALL "CEEDAYS" USING INDATE , PICSTR , LILDATE , DAYSFC.

           CALL "CEEDYWK" USING LILDATE , DAYOFWK , DYWKFC.

           COMPUTE FRIDAY = LILDATE + ADJUST ( DAYOFWK + 1 ).

           CALL "CEEDATE" USING FRIDAY , PICSTR , WKENDNG , DATEFC.
```

Figure 94. COBOL/370 example illustrating calls to CEEDAYS, CEEDYWK, and CEEDATE

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>
#define SUCCESS "\0\0\0\0"

void nmonths (_VSTRING *date, _VSTRING *pic, int n, _VSTRING *newdate) {

    _FLOAT8 secl;
    _FEEDBACK fc;
    _INT4 yy, mm, dd, hh, mi, ss, msec, yyinc;
    _CHAR80 tempdate;

    /* convert the input date to Lilian format */
    CEESECS(date,pic,&secl,&fc);

    if (memcmp(&fc,SUCCESS,4) != 0) {
       printf("CEESECS failed with message %d\n",fc.tok_msgno);
       newdate = NULL;
       return;
    }

    /* split the lilian date into its parts */
    CEESECI(&secl,&yy,&mm,&dd,&hh,&mi,&ss,&msec,&fc);

    if (memcmp(&fc,SUCCESS,4) != 0) {
       printf("CEESECI failed with message %d\n",fc.tok_msgno);
       newdate = NULL;
       return;
    }

    /* adjust the date n months */

    yyinc = n/12;
    mm = mm + (n - yyinc*12);
    yy = yy + yyinc + (mm-1)/12;
    if (mm > 12)
       mm = mm - 12;

    /* convert the date back to Lilian */
    CEEISEC(&yy,&mm,&dd,&hh,&mi,&ss,&msec,&secl,&fc);
```

Figure 93 (Part 1 of 2). C/370 example illustrating calls to CEESECS, CEESECI, CEEISEC, and CEEDATM

Figure 92 on page 400 and Figure 93 on page 401 contain COBOL/370 and C/370 examples, respectively, which illustrate how CEESECS, CEESECI, CEEISEC, and CEEDATM can be used in a routine that calculates the date *n* months from the input date.

```
          MOVE "1988/08/26 19:55:00" to TIMESTP-STRING.
          MOVE 50 TO TIMESTP-LENGTH.
          MOVE "YYYY/MM/DD HH:MI:SS" TO PICSTR-STRING.
          MOVE 50 TO PICSTR-LENGTH.

          CALL "CEESECS" USING TIMESTP , PICSTR , SECS , FC.
          IF FC = LOW-VALUE
             THEN DISPLAY "+->PASS CE1DT050_1"
             ELSE DISPLAY "-->FAIL CE1DT050_1"
          END-IF.

          CALL "CEESECI" USING SECS , YEAR , MONTH , DAYS , HOURS ,
                               MINUTES , SECONDS , MILLSEC , FC.
          IF FC = LOW-VALUE
             THEN DISPLAY "+->PASS CE1DT050_2"
             ELSE DISPLAY "-->FAIL CE1DT050_2"
          END-IF.

      * CONVERT TO ABSOLUTE MONTH NUMBER. ADD N MONTHS TO GET NEW
      * ABSOLUTE MONTH NUM. SUBTRACT 1 TO MAKE FOLLOWING CALCULATIONS
      * GIVE CORRECT RESULT.
          COMPUTE MONTHNUM = YEAR * 12 + MONTH + N - 1.

      * CONVERT MONTH NUM TO YEAR, NORMALIZE MONTH NUM TO MONTH 1 - 12
          DIVIDE MONTHNUM BY 12 GIVING YEAR REMAINDER MONTH.
          COMPUTE MONTH = MONTH + 1.

          CALL "CEEISEC" USING YEAR , MONTH , DAYS , HOURS , MINUTES ,
                               SECONDS , MILLSEC , SECS , FC.
          IF FC = LOW-VALUE
             THEN DISPLAY "+->PASS CE1DT050_3"
             ELSE DISPLAY "-->FAIL CE1DT050_3"
          END-IF.

          CALL "CEEDATM" USING SECS , PICSTR, TIMESTP, FC.
          IF FC = LOW-VALUE
             THEN DISPLAY "+->PASS CE1DT050_4"
             ELSE DISPLAY "-->FAIL CE1DT050_4"
          END-IF.
```

Figure 92. COBOL/370 example illustrating calls to CEESECS, CEESECI, CEEISEC, and CEEDATM

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#define SUCCESS "\0\0\0\0"

int hourdiff (_VSTRING *time1, _VSTRING *pic1,
              _VSTRING *time2, _VSTRING *pic2) {
    _FLOAT8 sec1,sec2;
    _FEEDBACK fc;

    /* convert the input date to Lilian format */
    CEESECS(time1,pic1,&sec1,&fc);

    if (memcmp(&fc,SUCCESS,4) != 0) {
       printf("CEESECS failed with message %d\n",fc.tok_msgno);
       return (-1);
    }

    CEESECS(time2,pic2,&sec2,&fc);

    if (memcmp(&fc,SUCCESS,4) != 0) {
       printf("CEESECS failed with message %d\n",fc.tok_msgno);
       return (-1);
    }

    /* calculate the difference */
    return (fabs(sec1 - sec2)/3600);
}
```

Figure 90. C/370 example illustrating calls to CEESECS

Figure 91 contains a COBOL/370 example illustrating how to convert a timestamp to number of seconds, then calculate the date and time 36 hours ago.

```
           MOVE "1988/07/26 19:55:00" to TIMESTP-STRING.
           MOVE 50 TO TIMESTP-LENGTH.
           MOVE "YYYY/MM/DD HH:MI:SS" TO PICSTR-STRING.
           MOVE 50 TO PICSTR-LENGTH.

           CALL "CEESECS" USING TIMESTP , PICSTR , SECONDS , FC.
           IF FC = LOW-VALUE
              THEN DISPLAY "++>PASS CE1DT020_1"
              ELSE DISPLAY "-->FAIL CE1DT020_1"
           END-IF.

           COMPUTE SECONDS = SECONDS - 36 * 60 * 60.

           CALL "CEEDATM" USING SECONDS , PICSTR , NEWTIME , FC.
           IF FC = LOW-VALUE
              THEN DISPLAY "++>PASS CE1DT020_2"
              ELSE DISPLAY "-->FAIL CE1DT020_2"
           END-IF.
```

Figure 91. COBOL/370 example illustrating calls to CEESECS and CEEDATM

```
MOVE "19880516190001" to TIMESTP-STRING.
MOVE 50 TO TIMESTP-LENGTH.
MOVE "YYYYMMDDHHMISS" TO PICSTR-STRING.
MOVE 50 TO PICSTR-LENGTH.

CALL "CEESECS" USING TIMESTP , PICSTR , SECOND1 , FC.
IF FC = LOW-VALUE
   THEN DISPLAY "++>PASS CE1DT020_1"
   ELSE DISPLAY "-->FAIL CE1DT020_1"
END-IF.

MOVE "1988-05-17-03.00.01" to TIMESTP-STRING.
MOVE 50 TO TIMESTP-LENGTH.
MOVE "YYYY-MM-DD-HH.MI.SS" TO PICSTR-STRING.
MOVE 50 TO PICSTR-LENGTH.

CALL "CEESECS" USING TIMESTP , PICSTR , SECOND2 , FC.
IF FC = LOW-VALUE
   THEN DISPLAY "++>PASS CE1DT020_2"
   ELSE DISPLAY "-->FAIL CE1DT020_2"
END-IF.

COMPUTE DIFF = (SECOND2 - SECOND1) / 3600.
```

Figure 89. COBOL/370 example illustrating calls to CEESECS

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>
#define SUCCESS "\0\0\0\0"

void future60 (_VSTRING *date, _VSTRING *picture,_VSTRING *newdate) {
    _INT4 lilian;
    _FEEDBACK fc;
    _CHAR80 tempdate;

    /* convert the input date to Lilian format */
    CEEDAYS(date,picture,&lilian,&fc);

    if (memcmp(&fc,SUCCESS,4) != 0) {
       printf("CEEDAYS failed with message %d\n",fc.tok_msgno);
       newdate = NULL;
       return;
    }

    /* make the date 60 days in the future */
    lilian = lilian + 60;

    /* reconvert the Lilian date to the format of the input */
    CEEDATE(&lilian,picture,&tempdate,&fc);

    if (memcmp(&fc,SUCCESS,4) != 0) {
       printf("CEEDATE failed with message %d\n",fc.tok_msgno);
       newdate = NULL;
       return;
    }
    memcpy(newdate->string,tempdate,picture->length);
    newdate->length = picture->length;
    return;
}
```

Figure 88. C/370 example illustrating calls to CEEDAYS and CEEDATE

Figure 89 on page 398 and Figure 90 on page 399 contains COBOL/370 and C/370 examples which illustrate how to use CEESECS to calculate in hours the difference between two timestamps.

```
MOVE "880516" to CHRDATE-STRING.
MOVE LENGTH OF CHRDATE-STRING TO CHRDATE-LENGTH.
MOVE "YYMMDD" TO PICSTR-STRING.
MOVE LENGTH OF PICSTR-STRING TO PICSTR-LENGTH.

CALL "CEEDAYS" USING CHRDATE , PICSTR , LILIAN , FC.
IF FC = LOW-VALUE
   THEN DISPLAY "++>PASS CE1DT010_1"
   ELSE DISPLAY "-->FAIL CE1DT010_1"
END-IF.

COMPUTE LILIAN = LILIAN + 60.
MOVE 50 TO PICSTR-LENGTH.

CALL "CEEDATE" USING LILIAN , PICSTR , NEWDATE , FC.
IF FC = LOW-VALUE
   THEN DISPLAY "++>PASS CE1DT010_2"
   ELSE DISPLAY "-->FAIL CE1DT010_2"
END-IF.
```

Figure 87. COBOL/370 example illustrating calls to CEEDAYS and CEEDATE

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _FEEDBACK fc;
   _FLOAT8 seconds1, seconds2;
   _VSTRING date,date_pic;
   #define SUCCESS "\0\0\0\0"

   /* use CEESECS to convert to seconds timestamp */
   strcpy(date.string,"09/13/91 23:23:23");
   date.length = strlen(date.string);
   strcpy(date_pic.string,"MM/DD/YY HH:MI:SS");
   date_pic.length = strlen(date_pic.string);

   CEESECS(&date,&date_pic,&seconds1,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEESECS failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }

   strcpy(date.string,"December 15, 1992 at 8:23:45 AM");
   date.length = strlen(date.string);
   strcpy(date_pic.string,"Mmmmmmmmmmmmz DD, YYYY at ZH:MI:SS AP");
   date_pic.length = strlen(date_pic.string);

   CEESECS(&date,&date_pic,&seconds2,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEESECS failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }

   printf("The number of seconds between: \n");
   printf("September 13, 1991 at 11:23:23 PM and December 15, 1992 at");
   printf(" 8:23:45 AM is:\n");
   printf("%f\n",seconds2 - seconds1);
}
```

CEEUTC — Get Coordinated Universal Time

CEEUTC is an alias of CEEGMT

Examples Using Several Date and Time Services

Figure 87 on page 396 contains a COBOL/370 example illustrating how to convert a date to Lilian format, then use the result to calculate a date 60 days in the future. Figure 88 on page 397 contains a C/370 example which illustrates how CEEDAYS and CEEDATE can be used in a routine that calculates the date 60 days from the input date. The output date is returned according to the same picture string as the input date.

Examples

1. COBOL/370 Example —

```
01   TIMESTP.
    05 TIMESTP-LENGTH PIC S9(4) COMP.
    05 TIMESTP-STRING PIC X(300).
01   PICSTR.
    05 PICSTR-LENGTH PIC S9(4) COMP.
    05 PICSTR-STRING PIC X(300).
77   SECONDS COMP-2.
77   FC PIC X(12).
:

     MOVE "Date: 1 January 2000" to TIMESTP-STRING.
     MOVE 50 TO TIMESTP-LENGTH.
     MOVE "Date: ZD Mmmmmmmmmmmmmmmz YYYY" TO PICSTR-STRING.
     MOVE 50 TO PICSTR-LENGTH.
     CALL "CEESECS" USING TIMESTP , PICSTR , SECONDS , FC.
```

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EB	3	2507	Insufficient data was passed to CEEDAYS or CEESECS. The Lilian value was not calculated.
CEE2EC	3	2508	The date value passed to CEEDAYS or CEESECS was invalid.
CEE2ED	3	2509	The Japanese or Republic of China Era passed to CEEDAYS or CEESECS was not recognized.
CEE2EE	3	2510	The hours value was not recognized.
CEE2EH	3	2513	The input date was not within the supported range.
CEE2EK	3	2516	The minutes value was not recognized.
CEE2EL	3	2517	The month value was not recognized.
CEE2EM	3	2518	An invalid picture string was specified.
CEE2EN	3	2519	The seconds value was not recognized.
CEE2EP	3	2521	The Japanese or Chinese year-within-Era value passed to CEEDAYS or CEESECS was zero.
CEE2ET	3	2525	CEESECS detected non-numeric data in a numeric field, or the timestamp string did not match the picture string.

Usage Notes

1. The inverse of CEESECS is CEEDATM, converting *output_seconds* to character format.
2. By default, 2-digit years are assumed to lie within the 100 year range starting 80 years prior to the system date. Thus, in 1990, all 2-digit years are assumed to represent dates between 1910 and 2009, inclusive. This range can be changed by using the callable service "CEESCEN — Set the Century Window" on page 387.
3. Elapsed time calculations can be performed easily on the *output_seconds*, since it represents elapsed time. Leap year and end-of-year anomalies are avoided.
4. If *picture_string* includes a Japanese Era symbol <JJJJ>, the YY position in *input_timestamp* represents the year number within the Japanese Era. For example, the year 1988 equals the Japanese year 63 in the Showa era. See Table 50 on page 367 for an additional example. Also see Table 51 on page 367 for a list of Japanese Eras supported by CEEDATE.
5. If *picture_string* includes an ROC Era symbol <CCCC> or <CCCCCCCC>, the YY position in *input_timestamp* represents the year number within the ROC (Republic of China) Era. For example, the year 1988 equals the ROC year 77 in the MinKow Era. See Table 50 on page 367 for an additional example. See Table 52 on page 367 for a list of ROC Eras supported by CEEDATE.

remaining values. For example:

```
1988-05-17-19:02 is equivalent to 1988-05-17-19:02:00
1988-05-17       is equivalent to 1988-05-17-00:00:00
```

picture_string (input)

a character string indicating the format of the date or timestamp value that you specified in *input_timestamp*.

Each character in the *picture_string* represents a character in *input_timestamp*. For example, if you specify MMDDYY HH.MI.SS as the *picture_string*, CEESECS reads an *input_char_date* of 060288 15.35.02 as 3:35:02 PM on 02 June 1988. If delimiters such as the slash (/) appear in the picture string, then leading zeros may be omitted. For example, the following calls to CEESECS:

```
CALL CEESECS('88/06/03 15.35.03', 'YY/MM/DD HH.MI.SS', fc, secs);
CALL CEESECS('88/6/3 15.35.03'  , 'YY/MM/DD HH.MI.SS', fc, secs);
CALL CEESECS('88/6/3 3.35.03 PM', 'YY/MM/DD HH.MI.SS AP', fc, secs);
CALL CEESECS('88.155 3.35.03 pm', 'YY.DDD HH.MI.SS AP', fc, secs);
```

all assign the same value to variable *secs*.

If picture string is left null or blank, CEESECS obtains *picture_string* based on the current value of the COUNTRY run-time option (see "COUNTRY" on page 224). For example, if the current value of the COUNTRY run-time option is FR (France), the date format would be DD.MM.YYYY.

See Table 49 on page 365 for a list of valid picture characters, and Table 50 on page 367 for examples of valid picture strings.

output_seconds (output)

a 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second 86,401 (24*60*60 + 01) in the Lilian format. 19:00:01.12 on 16 May 1988 is second 12,799,191,601.12.

The largest value that can be represented is 23:59:59.999 on 31 December 9999, which is second 265,621,679,999.999 in the Lilian format.

Note: A 64-bit double floating-point value can accurately represent approximately 16 significant decimal digits without loss of precision. Therefore, accuracy is available to the nearest millisecond (15 decimal digits).

If *input_timestamp* does not contain a valid date or timestamp, *output_seconds* is set to 0 and CEESECS terminates with non-zero feedback code.

fc (output)

an optional 12-byte feedback code passed by reference indicating the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main(void) {

  _INT4 year, month, day, hours, minutes, seconds, millisecs;
  _FLOAT8 input;
  _FEEDBACK fc;
  #define SUCCESS "\0\0\0\0"

  input = 13166064000.0;
  CEESECI(&input,&year,&month,&day,&hours,&minutes,&seconds,&millisecs,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEESECI failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("%f seconds corresponds to the date %d:%d:%d.%d %d/%d/%d\n",
         input,hours,minutes,seconds,millisecs,month,day,year);
}
```

CEESECS — Converts Timestamp to Number of Seconds

The CEESECS callable service converts a string representing a timestamp into a number representing the number of seconds since 00:00:00 14 October 1582. This service makes it easier to do time arithmetic, such as calculating the elapsed time between two timestamps.

Syntax

```
▸▸──CEESECS──(──input_timestamp──,──picture_string──,──────────▸

▸──output_seconds──,──┬────┬──)──────────────────────────────▸◂
                      └─fc─┘
```

input_timestamp (input)

a length-prefixed character string representing a date or timestamp in a format that matches that specified by *picture_string*.

The character string must contain between 5 and 80 picture characters, inclusive. *Input_timestamp* can contain leading or trailing blanks. Parsing begins with the first non-blank character (unless the picture string itself contains leading blanks. In this case, CEESECS skips exactly that many positions before parsing begins).

After a valid date is parsed, as determined by the format of the date which you specify in *picture_string*, all remaining characters are ignored by CEESECS. Valid dates are in the range between and including the dates 15 October 1582 to 31 December 9999. A full date must be specified. Valid times are in the range 00:00:00.000 to 23:59:59.999.

If any part or all of the time value is omitted, zeros are substituted for the

fc (output)

an optional 12-byte feedback code passed by reference indicating the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E9	3	2505	The number-of-seconds value was not within the supported range.

Usage Notes

1. The inverse of CEESECI is CEEISEC, which converts integer year, month, day, hour, second, and millisecond to number of seconds.
2. If the input value is a Lilian date instead of seconds, multiply the Lilian date by 86,400 (number of seconds in a day), and pass the new value to CEESECI.

Examples

1. COBOL/370 Example —

```
77   INSECS COMP-2.
77   YEAR PIC S9(9) COMP.
77   MONTH PIC S9(9) COMP.
77   DAYS PIC S9(9) COMP.
77   HOURS PIC S9(9) COMP.
77   MINUTES PIC S9(9) COMP.
77   SECONDS PIC S9(9) COMP.
77   MILLSEC PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE 13166064060 TO INSECS.
     CALL "CEESECI" USING INSECS , YEAR , MONTH , DAYS , HOURS ,
     MINUTES , SECONDS , MILLSEC , FC.
```

CEESECI — Convert Seconds to Integers

The CEESECI callable service converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond. Use CEESECI instead of CEEDATM when the output is needed in numeric format rather than character format.

Syntax

```
▸▸──CEESECI─(─input_seconds─,─output_year─,──output_month─,─▸

▸──output_day─,─output_hours─,─output_minutes─,───────────▸

▸──output_seconds─,─output_milliseconds─,─┬────┬──────────▸◂
                                          └─fc─┘
```

input_seconds

a 64-bit double floating point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds.

For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range for *input_seconds* is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999).

If *input_seconds* is invalid, all output parameters except the feedback code are set to zero.

output_year (output)

a 32-bit binary integer representing the year.

The range of valid *output_years* is 1582 to 9999, inclusive.

output_month (output)

a 32-bit binary integer representing the month.

The range of valid *output_months* is 1 to 12.

output_day (output)

a 32-bit binary integer representing the day.

The range of valid *output_days* is 1 to 31.

output_hours (output)

a 32-bit binary integer representing the hour.

The range of valid *output_hours* is 0 to 23.

output_minutes (output)

a 32-bit binary integer representing the minutes.

The range of valid *output_minutes* is 0 to 59.

output_seconds (output)

a 32-bit binary integer representing the seconds.

The range of valid *output_seconds* is 0 to 59.

output_milliseconds (output)

a 32-bit binary integer representing milliseconds.

The range of valid *output_milliseconds* is 0 to 999.

fc (output)

an optional 12-byte feedback code passed by reference indicating the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E6	3	2502	The UTC/GMT was not available from the system.

Usage Note

1. Century intervals are kept as thread-level data, so changing the interval in one thread does not affect the interval in another thread.

Examples

1. COBOL/370 Example —

```
77   STARTCW PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE 80 TO STARTCW.
     CALL "CEESCEN" USING STARTCW , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main (void) {

  _INT4 century_start;
  _FEEDBACK fc;
  #define SUCCESS "\0\0\0\0"

  century_start = 50;

  CEESCEN(&century_start,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEESCEN failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
}
```

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main (void) {

  _INT4 century_start;
  _FEEDBACK fc;
  #define SUCCESS "\0\0\0\0"

  /* query the century window */
  CEEQCEN(&century_start,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEQCEN failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }

  /* if the century window is not 50 set it to 50 */
  if (century_start != 50) {
     century_start = 50;

     CEESCEN(&century_start,&fc);
     if (memcmp(&fc,SUCCESS,4) != 0) {
        printf("CEESCEN failed with message number %d\n",fc.tok_msgno);
        exit(2999);
     }
  }
}
```

CEESCEN — Set the Century Window

The CEESCEN callable service sets the century in which LE/370 assumes 2-digit year values lie. Use it in conjunction with CEEDAYS ("CEEDAYS — Convert Date to Lilian Format" on page 361) or CEESECS ("CEESECS — Converts Timestamp to Number of Seconds" on page 391) when:

- you process date values that contain 2-digit years (for example, in the YYMMDD format)
- when the LE/370 default century interval does not meet the requirements of a particular application.

Syntax

```
▶▶──CEESCEN──(──century_start──,──┬────┬──)──────────────────▶◀
                                  └─fc─┘
```

century_start

an integer between 0 and 100 that sets the century window.

A value of 80 (the LE/370 default), for example, places all two-digit years within the 100 year window starting 80 years before the system date. In 1990, therefore, all two-digit years are assumed to represent dates between 1910 and 2009, inclusive.

For example, if the LE/370 default is in effect all 2-digit years are assumed to lie within the 100-year window starting 80 years prior to the system date. CEEQCEN would then return the value "80". A value of 80 indicates that, in 1990, LE/370 assumes that all two-digit years lie within the 100 year window starting 80 years before the system date (between 1910 and 2009, inclusive).

fc (output)
an optional 12-byte feedback code passed by reference indicating the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Examples

1. COBOL/370 Example —

```
77   STARTCW PIC S9(9) COMP.
77   FC PIC X(12).
:

     CALL "CEEQCEN" USING STARTCW , FC.
```

Examples

1. COBOL/370 Example —

```
77   LILIAN PIC S9(9) COMP.
77   SECONDS COMP-2.
77   GREGORN PIC X(17).
77   FC PIC X(12).
:

     CALL "CEELOCT" USING LILIAN , SECONDS , GREGORN , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main(void) {

  _FEEDBACK fc;
  _INT4     lil_date;
  _FLOAT8   local_date;
  _CHAR17   gregorian_date;
  #define SUCCESS "\0\0\0\0"

  CEELOCT(&lil_date,&local_date,gregorian_date,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEELOCT failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }

  printf("The current date is YYYYMMDDHHMISS999\n");
  printf("                    %.17s\n",gregorian_date);
}
```

CEEQCEN — Query the Century Window

The CEEQCEN callable service queries the century within which LE/370 assumes 2-digit year values lie. Use it in conjunction with the CEESCEN callable service ("CEESCEN — Set the Century Window" on page 387) when it is necessary to save and restore the current setting.

Syntax

```
▸▸——CEEQCEN—(—century_start—,—┬────┬—)——————————————————▸◂
                              └─fc─┘
```

century_start

an integer between 0 and 100 returned by CEEQCEN, giving the year upon which the century window is based.

output_Lilian (output)
a 32-bit binary integer representing the current *local* date in the Lilian format, that is, day 1 = 15 October 1582, day 148,887 = 4 June 1990.

If the local time is not available from the system, *output_Lilian* is set to 0 and CEELOCT terminates with a nonzero feedback code.

output_seconds (output)
a 64-bit double-floating point number representing the current *local* date and time as the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). 19:00:01.078 on 4 June 1990 is second number 12,863,905,201.078.

If the local time is not available from the system, *output_seconds* is set to 0 and CEELOCT terminates with a nonzero feedback code.

output_Gregorian (output)
a 17-byte fixed length character string in the form YYYYMMDDHHMISS999 representing local year, month, day, hour, minute, second, and millisecond.

fc (output/optional)
an optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E6	3	2502	The UTC/GMT was not available from the system.

Usage Notes

1. You can use the CEEGMT callable service ("CEEGMT — Get Current Greenwich Mean Time" on page 376) to determine Greenwich Mean Time (GMT).
2. You can use the CEEGMTO callable service ("CEEGMTO — Get Offset From Greenwich Mean Time to Local Time" on page 378) to obtain the offset from GMT to local time.
3. If the format of output_Gregorian is inappropriate, you can use the CEEDATM callable service ("CEEDATM — Convert Seconds to Character Timestamp" on page 371) to convert *output_seconds* to any required format.
4. The character value returned by CEELOCT is designed to match that produced by existing language intrinsic functions. The numeric values returned can be used to simplify date calculations.
5. CICS Consideration — CEELOCT does not use the OS TIME macro. Therefore, CEELOCT can be used under CICS.

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main(void) {

  _INT4 year, month, day, hours, minutes, seconds, millisecs;
  _FLOAT8 output;
  _FEEDBACK fc;
  #define SUCCESS "\0\0\0\0"

  year = 1991;
  month = 9;
  day = 13;
  hours = 4;
  minutes = 34;
  seconds = 25;
  millisecs = 746;

  CEEISEC(&year,&month,&day,&hours,&minutes,&seconds,&millisecs,&output,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEISEC failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("The number of seconds between 00:00:00.00 10/14/1582 and");
  printf(" 04:34:25.746 09/13/1991 is %.3f\n",output);
}
```

CEELOCT — Get Current Local Time

The CEELOCT callable service returns the current Local Time in three formats:

- Lilian date (the number of days since 14 October 1582)
- Lilian timestamp (the number of seconds since 00:00:00 14 October 1582)
- Gregorian character string (in the form YYYYMMDDHHMISS999).

These values are compatible with other LE/370 date and time services, and with existing language intrinsic functions. This service can be duplicated by calling the CEEGMT, CEEGMTO, and CEEDATM date and time services in succession. However, CEELOCT performs the same service with much greater speed.

Syntax

```
►►──CEELOCT──(──output_Lilian──,──output_seconds──,──────────►

►──output_Gregorian──,──┬────┬──)─────────────────────────►◄
                        └─fc─┘
```

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EE	3	2510	The hours value was not recognized.
CEE2EF	3	2511	The day value was invalid for year and month specified.
CEE2EH	3	2513	The input date was not within the supported range.
CEE2EI	3	2514	The year value was not within the supported range.
CEE2EJ	3	2515	The milliseconds value was not recognized.
CEE2EK	3	2516	The minutes value was not recognized.
CEE2EL	3	2517	The month value was not recognized.
CEE2EN	3	2519	The seconds value was not recognized.

Usage Notes

1. The inverse of CEEISEC is CEESECI, which converts number of seconds to integer year, month, day, hour, minute, second, and millisecond.
2. To convert *output_seconds* to a Lilian day number, divide *output_seconds* by 86,400 (the number of seconds in a day).

Examples

1. COBOL/370 Example —

```
77   YEAR PIC S9(9) COMP.
77   MONTH PIC S9(9) COMP.
77   DAYS PIC S9(9) COMP.
77   HOURS PIC S9(9) COMP.
77   MINUTES PIC S9(9) COMP.
77   SECONDS PIC S9(9) COMP.
77   MILLSEC PIC S9(9) COMP.
77   OUTSECS COMP-2.
77   FC PIC X(12).
 :

     MOVE 2000 TO YEAR.
     MOVE 1 TO MONTH.
     MOVE 1 TO DAYS.
     MOVE 0 TO HOURS.
     MOVE 0 TO MINUTES.
     MOVE 0 TO SECONDS.
     MOVE 0 TO MILLSEC.
     CALL "CEEISEC" USING YEAR , MONTH , DAYS , HOURS , MINUTES ,
     SECONDS , MILLSEC , OUTSECS , FC.
```

input_year (input)
a 32-bit binary integer representing the year.

The range of valid *input_years* is 1582 to 9999, inclusive.

input_month (input)
a 32-bit binary integer representing the month.

The range of valid *input_months* is 1 to 12.

input_day (input)
a 32-bit binary integer representing the day.

The range of valid *input_days* is 1 to 31.

input_hours (input)
a 32-bit binary integer representing the hours.

The range of valid *input_hours* is 0 to 23.

input_minutes (input)
a 32-bit binary integer representing the minutes.

The range of valid *input_minutes* is 0 to 59.

input_seconds (input)
a 32-bit binary integer representing the seconds.

The range of valid *input_seconds* is 0 to 59.

input_milliseconds (input)
a 32-bit binary integer representing milliseconds.

The range of valid *input_milliseconds* is 0 to 999.

output_seconds (output)
a 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds.

For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range of *output_seconds* is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999).

If any input values are invalid, *output_seconds* is set to zero.

fc an optional 12-byte feedback code passed by reference indicating the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

Examples

1. COBOL/370 Example —

```
77   HOURS PIC S9(9) COMP.
77   MINUTES PIC S9(9) COMP.
77   SECONDS COMP-2.
77   FC PIC X(12).
:

     CALL "CEEGMTO" USING HOURS , MINUTES , SECONDS , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main(void) {

  _FEEDBACK fc;
  _INT4     GMT_hours,GMT_mins;
  _FLOAT8   GMT_secs;
  #define SUCCESS "\0\0\0\0"

  CEEGMTO(&GMT_hours,&GMT_mins,&GMT_secs,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEGMTO failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("The difference between GMT and the local time is:\n");
  printf("%d hours, %d minutes\n",GMT_hours,GMT_mins);
}
```

CEEISEC — Convert Integers to Seconds

The CEEISEC callable service converts separate binary integers representing year, month, day, hour, minute, second, and millisecond to a number representing the number of seconds since 00:00:00 14 October 1582. Use CEEISEC instead of CEESECS when the input is in numeric format rather than character format.

Syntax

```
▸▸──CEEISEC──(─input_year─,─input_month─,──input_day─,────────▸

▸───input_hours─,─input_minutes─,─input_seconds─,─────────────▸

▸───input_milliseconds─,─output_seconds─,─┬────┬─)──────────▸◂
                                          └─fc─┘
```

If local time offset is not available, *offset_hours* = 0 and CEEGMTO terminates with a non-zero feedback code.

offset_minutes (output)
a 32-bit binary integer representing the number of additional minutes that local time is ahead of or behind GMT.

The range of *offset_minutes* is 0 to 59.

If the local time offset is not available, *offset_minutes* = 0 and CEEGMTO terminates with non-zero feedback code.

offset_seconds (output)
a 64-bit double floating-point number representing the offset from GMT to local time, in seconds.

For example, Pacific Standard Time is eight hours behind GMT. If local time is in the Pacific time zone during standard time, CEEGMTO would return –28,800 (–8 * 60 * 60). The range of *offset_seconds* is –43,200 to +46,800. *Offset_seconds* can be used with CEEGMT to calculate local date and time. See "CEEGMT — Get Current Greenwich Mean Time" on page 376 for more information.

If the local time offset is not available from the system, *offset_seconds* is set to 0 and CEEGMTO terminates with a non-zero feedback code.

fc (output)
an optional 12-byte feedback code passed by reference indicating the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E7	3	2503	The offset from UTC/GMT to local time was not available from the system.

Usage Notes

1. The CEEDATM service can be used to convert number of seconds to a character timestamp. See "CEEDATM — Convert Seconds to Character Timestamp" on page 371 for more information.
2. CEEGMTO does not use the OS TIME macro. Therefore, CEEGMTO works under CICS.

Examples

1. COBOL/370 Example —

```
77   LILIAN PIC S9(9) COMP.
77   SECONDS COMP-2.
77   FC PIC X(12).
:

     CALL "CEEGMT" USING LILIAN , SECONDS , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main(void) {

  _FEEDBACK fc;
  _INT4     lilGMT_date;
  _FLOAT8   secGMT_date;
  #define SUCCESS "\0\0\0\0"

  CEEGMT(&lilGMT_date,&secGMT_date,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEGMT failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("The current Lilian date in Greenwich,England is %d\n",
         lilGMT_date);
}
```

CEEGMTO — Get Offset From Greenwich Mean Time to Local Time

The CEEGMTO callable service returns values to the calling routine that represent the difference between the local system time and Greenwich Mean Time.

Syntax

```
▸▸──CEEGMTO──(──offset_hours──,──offset_minutes──,──────────────▸

▸──offset_seconds──,──┬────┬──)──────────────────────────────▸◂
                      └─fc─┘
```

offset_hours (output)

a 32-bit binary integer representing the offset from GMT to local time, in hours.

For example, for Pacific Standard Time, *offset_hours* = –8.

The range of *offset_hours* is –12 to +13 (+13 = Daylight Savings Time in the +12 time zone).

output_GMT_Lilian (output)
a 32-bit binary integer representing the current date in Greenwich, England, in the Lilian format (the number of days since 14 October 1582). This date is also known as Coordinated Universal Time (UTC).

For example, 16 May 1988 is day number 148138. If GMT is not available from the system, *output_GMT_Lilian* is set to 0 and CEEGMT terminates with a non-zero feedback code.

output_GMT_seconds (output)
a 64-bit double floating-point number representing the current date and time in Greenwich, England as the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds.

For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). 19:00:01.078 on 16 May 1988 is second number 12,799,191,601.078. If GMT is not available from the system, *output_GMT_seconds* is set to 0 and CEEGMT terminates with a non-zero feedback code.

fc (output)
an optional 12-byte feedback code passed by reference indicating the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions may result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E6	3	2502	The UTC/GMT was not available from the system.

Usage Notes

1. The CEEGMT service assumes that your system's TOD (time-of-day) clock is set to Greenwich Mean Time and is based on the standard epoch. If this is not the case, the results of this service will not be meaningful.
2. Use CEEGMTO ("CEEGMTO — Get Offset From Greenwich Mean Time to Local Time" on page 378) to obtain the offset from GMT to local time.
3. The values returned by CEEGMT are handy for elapsed time calculations. For example, you can calculate the time elapsed between two calls to CEEGMT by calculating the differences between the returned values.
4. CEEDATE converts *output_GMT_Lilian* to a character date, and CEEDATM converts *output_GMT_seconds* to a character timestamp.
5. CICS Considerations — CEEGMT does not use the OS TIME macro. Therefore, CEEGMT can be used under CICS.

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main (void) {

  _INT4 in_date, day;
  _FEEDBACK fc;
  #define SUCCESS "\0\0\0\0"

  in_date = 139370;

  CEEDYWK(&in_date,&day,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEDYWK failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("Lilian date %d, occurs on a ",in_date);
  switch(day) {
     case 1: printf("Sunday.\n");
             break;
     case 2: printf("Monday.\n");
             break;
     case 3: printf("Tuesday.\n");
             break;
     case 4: printf("Wednesday.\n");
             break;
     case 5: printf("Thursday.\n");
             break;
     case 6: printf("Friday.\n");
             break;
     case 7: printf("Saturday.\n");
             break;
  }
}
```

CEEGMT — Get Current Greenwich Mean Time

The CEEGMT callable service returns the current Greenwich Mean Time (GMT) as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582. These values are compatible with those generated and used by the other LE/370 date and time services.

Syntax

```
▸▸——CEEGMT—(—output_GMT_Lilian—,—output_GMT_seconds—,——————▸

▸——————————)————————————————————————————————————————————————▸◂
   └─fc─┘
```

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EG	3	2512	The Lilian date value was not within the supported range.

Examples

1. COBOL/370 Example —

```
77   LILIAN PIC S9(9) COMP.
77   DAYNUM PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE 152385 TO LILIAN.
     CALL "CEEDYWK" USING LILIAN , DAYNUM , FC.
```

Table 54 (Page 2 of 2). Sample Output of CEEDATM

input_seconds	*picture_string*	*output_timestamp*
12,799,191,662.009	YYYY	1988
	YY	88
	Y	8
	MM	05
	ZM	5
	RRRR	V
	MMM	MAY
	Mmm	May
	Mmmmmmmmmm	Maybbbbbb
	Mmmmmmmmmz	May
	DD	16
	ZD	16
	DDD	137
	HH	19
	ZH	19
	MI	01
	SS	02
	99	00
	999	009
	AP	PM
	WWW	MON
	Www	Mon
	Wwwwwwwwww	Mondaybbbb
	Wwwwwwwwwz	Monday

CEEDYWK — Calculate Day of Week from Lilian Date

The CEEDYWK service calculates the day of the week on which a Lilian date falls. The day of the week is returned to the calling routine as a number between 1 and 7.

Syntax

```
▸▸——CEEDYWK—(—input_Lilian_date—,—output_day_no—,—┬────┬—)—;—▸◂
                                                  └─fc─┘
```

input_Lilian_date (input)
a 32-bit binary integer representing the Lilian date, the number of days since 14 October 1582.

For example, 16 May 1988 is day number 148138. The valid range of *input_Lilian_date* is between 1 and 3,074,324 (15 October 1582 and 31 December 9999).

output_day_no (output)
a 32-bit binary integer representing *input_Lilian_date*'s day-of-week: 1=Sunday, 2=Monday, ..., 7=Saturday.

If *input_Lilian_date* is invalid, *output_day_no* is set to 0 and CEEDYWK terminates with a non-zero feedback code.

fc (output)
an optional 12-byte feedback code passed by reference indicating the result of the service.

Examples

1. COBOL/370 Example —

```
77   SECONDS COMP-2.
    01   PICSTR.
    05 PICSTR-LENGTH PIC S9(4) COMP.
    05 PICSTR-STRING PIC X(300).
77   TIMESTP PIC X(80).
77   FC PIC X(12).
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _FEEDBACK fc;
   _FLOAT8 seconds = 12904183403.0;
   _VSTRING date,date_pic;
   _CHAR80 out_date;
   #define SUCCESS "\0\0\0\0"

   strcpy(date_pic.string,"Mmmmmmmmmmmz DD, YYYY at ZH:MI:SS AP");
   date_pic.length = strlen(date_pic.string);

   CEEDATM(&seconds,&date_pic,out_date,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEDATM failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }

   printf("%.80s\n",out_date);
}
```

Table 54 (Page 1 of 2). Sample Output of CEEDATM		
input_seconds	***picture_string***	***output_timestamp***
12,799,191,601.000	YYMMDD HH:MI:SS YY-MM-DD YYMMDDHHMISS YY-MM-DD HH:MI:SS YYYY-MM-DD HH:MI:SS AP	880516 19:00:01 88-05-16 880516190001 88-05-16 19:00:01 1988-05-16 07:00:01 PM
12,799,191,661.986	DD Mmm YY DD MMM YY HH:MM WWW, MMM DD, YYYY ZH:MI AP Wwwwwwwwwz, ZM/ZD/YY HH:MI:SS.99	16 May 88 16 MAY 88 19:01 MON, MAY 16, 1988 7:01 PM Monday, 5/16/88 19:01:01.98

See "COUNTRY" on page 224 for more information about the COUNTRY run-time option and "CEEFMDT — Obtain Default Date and Time Format" on page 341 for information about how to obtain a default timestamp for a given country code.

output_timestamp (output)

an 80-character string, the result of converting *input_seconds* to the format specified by *picture_string*.

If necessary, the output is truncated to the length of *output_timestamp*. See Table 54 on page 373 for sample output.

If *input_seconds* is invalid, *output_timestamp* is set to all blanks and CEEDATM terminates with a non-zero feedback code.

fc (output)

an optional 12-byte feedback code passed by reference indicating the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E9	3	2505	The number-of-seconds value was not within the supported range.
CEE2EA	3	2506	<JJJJ>, <CCCC> or <CCCCCCCC> was used in a picture string passed to CEEDATM, but the input number-of-seconds value was not within the supported range. The Era could not be determined.
CEE2EM	3	2518	An invalid picture string was specified.
CEE2EV	2	2527	The timestamp string returned by CEEDATM was truncated.
CEE3CF	2	3471	The country code *country-code* was invalid for CEEFMDT. The default date and time picture string *datetime-string* was returned.

Usage Notes

1. The inverse of CEEDATM is CEESECS, which converts timestamps to number of seconds.
2. If *picture_string* includes the Japanese Era symbol <JJJJ>, the YY position in *output_timestamp* represents "year within Japanese Era." See Table 50 on page 367 for an example. See Table 51 on page 367 for a list of Japanese Eras supported by CEEDATM.
3. If *picture_string* includes the ROC Era symbol <CCCC> or <CCCCCCCC>, the YY position in *output_timestamp* represents "year within ROC Era." See Table 50 on page 367 for an example. See Table 52 on page 367 for a list of ROC Eras supported by CEEDATM.

Table 53 (Page 2 of 2). Sample Output of CEEDATE		
input_Lilian_date	***picture_string***	***output_char_date***
148141	DDD YYDDD YY.DDD YYYY.DDD	140 88140 88.140 1988.140
148142	YY/MM/DD HH:MI:SS.99 YYYY/ZM/ZD ZH:MI AP	88/05/20 00:00:00.00 88/5/20 0:00 AM
148143	WWW., MMM DD, YYYY Www., Mmm DD, YYYY Wwwwwwwwww, Mmmmmmmmmm DD, YYYY Wwwwwwwwwz, Mmmmmmmmmz DD, YYYY	SAT., MAY 21, 1988 Sat., May 21, 1988 Saturdaybb, Maybbbbbbb 21, 1988 Saturday, May 21, 1988

CEEDATM — Convert Seconds to Character Timestamp

The CEEDATM callable service converts a number representing the number of seconds since 00:00:00 14 October 1582 to a character format. The format of the output is a character string, for example, "1988/07/26 20:37:00."

Syntax

```
▸▸──CEEDATM──(──input_seconds──,──picture_string──,──────────────▸

▸──output_timestamp──,──┬────┬──)─────────────────────────────▸◂
                        └─fc─┘
```

input_seconds (input)
a 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds.

For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range of *input_seconds* is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999).

picture_string (input)
a length-prefixed string representing the desired format of *output_timestamp*, for example MM/DD/YY HH:MI AP.

Each character in the *picture_string* represents a character in *output_timestamp*. If delimiters such as the slash (/) appear in the picture string, then they are copied as is to *output_timestamp*.

See Table 49 on page 365 for a list of valid picture characters, and Table 50 on page 367 for examples of valid picture strings.

If *picture_string* is null or blank, CEEDATM obtains *picture_string* based on the current value of the COUNTRY run-time option. For example, if the current value of the COUNTRY run-time option is US (United States), the date-time format would be "MM/DD/YY ZH:MI:SS AP" ; if the current COUNTRY value is FR (France), however, the date-time format would be "DD.MM.YYYY HH:MI:SS."

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _FEEDBACK fc;
   _INT4 lil_date = 139370;   /* May 14, 1964 */
   _VSTRING date_pic,date;
   _CHAR80 date_out;
   #define SUCCESS "\0\0\0\0"

   strcpy(date_pic.string,"The date is Wwwwwwwwwz, Mmmmmmmmmz ZD, YYYY");
   date_pic.length = strlen(date_pic.string);

   CEEDATE(&lil_date,&date_pic,date_out,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEDATE failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }
   printf("%.80s\n",date_out);
}
```

Table 53 (Page 1 of 2). Sample Output of CEEDATE

input_Lilian_date	***picture_string***	***output_char_date***
148138	YY YYMM YY-MM YYMMDD YYYYMMDD YYYY-MM-DD YYYY-ZM-ZD <JJJJ> YY.MM.DD <CCCC> YY.MM.DD	88 8805 88-05 880516 19880516 1988-05-16 1988-5-16 *Showa* 63.05.16 (in a DBCS string) *MinKow* 77.05.16 (in a DBCS string)
148139	MM MMDD MM/DD MMDDYY MM/DD/YYYY ZM/DD/YYYY	05 0517 05/17 051788 05/17/1988 5/17/1988
148140	DD DDMM DDMMYY DD.MM.YY DD.MM.YYYY DD Mmm YYYY	18 1805 180588 18.05.88 18.05.1988 18 May 1988

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EG	3	2512	The Lilian date value was not within the supported range.
CEE2EM	3	2518	An invalid picture string was specified.
CEE2EQ	3	2522	<JJJJ>, <CCCC> or <CCCCCCCC> was used in a picture string passed to CEEDATE, but the Lilian date value was not within the supported range. The Era could not be determined.
CEE2EU	2	2526	The date string returned by CEEDATE was truncated.

Usage Notes

1. The inverse of CEEDATE is CEEDAYS, which converts character dates to the Lilian format.
2. If *picture_string* includes a Japanese Era symbol <JJJJ>, the YY position in *output_char_date* is replaced by the year number within the Japanese Era. For example, the year 1988 equals the Japanese year 63 in the Showa era. See Table 50 on page 367 for an additional example. Also see Table 51 on page 367 for a list of Japanese Eras supported by CEEDATE.
3. If *picture_string* includes an ROC Era symbol <CCCC> or <CCCCCCCC>, the YY position in *output_char_date* is replaced by the year number within the ROC (Republic of China) Era. For example, the year 1988 equals the ROC year 77 in the MinKow Era. See Table 50 on page 367 for an additional example. Also see Table 52 on page 367 for the list of ROC Eras supported by CEEDATE.

Examples

1. COBOL/370 Example —

```
77   LILIAN PIC S9(9) COMP.
    01   PICSTR.
    05 PICSTR-LENGTH PIC S9(4) COMP.
    05 PICSTR-STRING PIC X(300).
77   CHRDATE PIC X(80).
77   FC PIC X(12).
:

     MOVE 152385 TO LILIAN.
     MOVE "Date: ZD Mmmmmmmmmmmmmmmz YYYY" TO PICSTR-STRING.
     MOVE 50 TO PICSTR-LENGTH.
     CALL "CEEDATE" USING LILIAN , PICSTR , CHRDATE , FC.
```

CEEDATE — Convert Lilian Date to Character Format

The CEEDATE service converts a number representing a Lilian date to a date written in character format. The output is a character string such as "1988/07/26."

Syntax

```
▶▶──CEEDATE──(──input_Lilian_date──,──picture_string──,──────────▶

▶──output_char_date──,──┬────┬──)────────────────────────────────▶◀
                        └─fc─┘
```

input_Lilian_date (input)

a 32-bit binary integer representing the Lilian date.

The Lilian date is the number of days since 14 October 1582. For example, 16 May 1988 is Lilian day number 148138.

The valid range of Lilian dates is 1 to 3,074,324 (15 October 1582 to 31 December 9999).

picture_string (input)

a length-prefixed string representing the desired format of *output_char_date*, for example MM/DD/YY. Each character in *picture_string* represents a character in *output_char_date*. If delimiters such as the slash (/) appear in the picture string, then they are copied as is to *output_char_date*.

See Table 49 on page 365 for a list of valid picture characters, and Table 50 on page 367 for examples of valid picture strings.

If *picture_string* is null or blank, CEEDATE obtains *picture_string* based on the current value of the COUNTRY run-time option. For example, if the current value of the COUNTRY run-time option is US (United States), the date format would be MM/DD/YY. If the current COUNTRY value is FR (France), the date format would be DD.MM.YYYY. This default mechanism makes it easy for translation centers to specify the preferred date, and for applications and library routines to use this format automatically. See "COUNTRY" on page 224 for more information about the COUNTRY run-time option and "CEEFMDA — Obtain Default Date Format" on page 338 for information about how to obtain the default format for a given country code.

output_char_date (output)

an 80-character string that is the result of converting *input_Lilian_date* to the format specified by *picture_string*.

See Table 53 on page 370 for sample output dates.

If *input_Lilian_date* is invalid, *output_char_date* is set to all blanks. CEEDATE terminates with a non-zero feedback code.

fc (output)

an optional 12-byte feedback code passed by reference.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

Table 50. Examples of Picture Terms Recognized by Run-time Date/Time Services		
Picture Terms	**Example**	**Notes**
YYMMDD YYYYMMDD YYYY-MM-DD <JJJJ> YY.MM.DD <CCCC> YY.MM.DD	880516 19880516 1988-05-16 *Showa* 63.05.16 *MinKow* 77.05.16	 1988-5-16 would also be valid input. *Showa* is a Japanese Era name. *Showa* 63 = 1988. *MinKow* is an ROC Era name. *MinKow* 77 = 1988.
MMDDYY MM/DD/YY ZM/ZD/YY MM/DD/YYYY MM/DD/Y	050688 05/06/88 5/6/88 05/06/1988 05/06/8	 1-digit year format (Y) valid for output only
DD.MM.YY DD-RRRR-YY DD MMM YY DD Mmmmmmmmmm YY ZD Mmmmmmmmmz YY Mmmmmmmmmz ZD, YYYY ZDMMMMMMMMMZZYY	09.06.88 09-VI -88 09 JUN 88 09 June 88 9 June 88 June 9, 1988 9JUNE88	 Z suppresses zeros/blanks
YY.DDD YYDDD YYYY/DDD	88.137 88137 1988/137	Julian date
YYMMDDHHMISS YYYYMMDDHHMISS YYYY-MM-DD HH:MI:SS.999 WWW, ZM/ZD/YY HH:MI AP Wwwwwwwwwz, DD Mmm YYYY, ZH:MI AP	880516204229 19880516204229 1988-05-16 20:42:29.046 MON, 5/16/88 08:42 PM Monday, 16 May 1988, 8:42 PM	Timestamp — valid only for CEESECS and CEEDATM. If used with CEEDATE, time positions are left blank. If used with CEEDAYS, HH, MI, SS, and 999 fields ignored.

Table 51. Japanese Eras used by Run-time Date/Time Services when <JJJJ> specified			
First date of Japanese Era	**Era Name**	**Era Name in IBM Japanese DBCS Code**	**Valid Year Values**
1868-09-08	Meiji	X'0E45A645840F'	01-45
1912-07-30	Taisho	X'0E455B45770F'	01-15
1926-12-25	Showa	X'0E45B3457A0F'	01-64
1989-01-08	Heisei	X'0E458D45BA0F'	01-999 (01 = 1989)

Table 52. Republic of China Eras used by run-time environment Date/Time Services when <CCCC>or <CCCCCCCC> specified			
First date of ROC Era	**Era Name**	**Era Name in Traditional Chinese DBCS Code**	**Valid Year (YY, ZYY) Values**
1912-01-01	*MinKow* *ChuHwaMinKow*	X'0E4D8256CE0F' X'0E4C845ADD4D8256CE0F'	01-999 (77 = 1988)

Table 49 (Page 2 of 2). Picture Terms Used in Picture Strings			
Picture Terms	**Explanation**	**Valid Values**	**Notes**
<JJJJ>	Japanese Era name in DBCS characters	*Heisei* (X'0E458D045BA0F') *Showa* (X'0E45B3457A0F') *Taisho* (X'0E455B45770F') *Meiji* (X'0E45A645840F')	Affects YY field: if <JJJJ> specified, YY means "year within Japanese Era," e.g., 1988 = Showa 63. See example in Table 50 on page 367.
<CCCC> <CCCCCCCC>	Republic of China (ROC) Era name in DBCS characters	*MinKow* (X'0E4D8256CE0F') *ChuHwaMinKow* (X'0E4C845ADD4D8256CE0F')	Affects YY field: if <CCCC> specified, YY means "year within ROC Era," e.g., 1988 = Minkow 77. See example in Table 50 on page 367.
MM ZM	2-digit month 1- or 2-digit month	01-12 1-12	For output, leading zero suppressed. For input, ZM treated as MM.
RRRR RRRZ	Roman numeral month	I-XII (Left justified)	For input source string is folded to uppercase. For output, uppercase only. I=Jan, II=Feb, ..., XII=Dec.
MMM Mmm MMMM...M Mmmm...m MMMMMMMMMZ Mmmmmmmmmz	3-char month, uppercase 3-char month, mixed case 3-20 char mo., uppercase 3-20 char mo., mixed case trailing blanks suppressed trailing blanks suppressed	JAN-DEC Jan-Dec JANUARYƀƀ-DECEMBERƀ Januaryƀƀ-Decemberƀ JANUARY-DECEMBER January-December	For input, source string always folded to uppercase. For output, M generates uppercase and m generates lowercase. Output is padded with blanks (ƀ) (unless Z specified) or truncated to match the number of Ms, up to 20.
DD ZD DDD	2-digit day of month 1- or 2-digit day of mo. day of year (Julian day)	01-31 1-31 001-366	For output, leading zero is always suppressed. For input, ZD treated as DD.
HH ZH	2-digit hour 1- or 2-digit hour	00-23 0-23	For output, leading zero suppressed. For input, ZH treated as HH. If AP specified, valid values are 01-12.
MI	minute	00-59	
SS	second	00-59	
9 99 999	tenths of a second hundredths of a second thousandths of a second	0-9 00-99 000-999	No rounding.
AP ap A.P. a.p.	AM/PM indicator	AM or PM am or pm A.M. or P.M. a.m. or p.m.	AP affects HH/ZH field. For input, source string always folded to uppercase. For output, AP generates uppercase and ap generates lowercase.
W WWW Www WWW...W Www...w WWWWWWWWWZ Wwwwwwwwwz	1-char day-of-week 3-char day, uppercase 3-char day, mixed case 3-20 char day, uppercase 3-20 char day, mixed case trailing blanks suppressed trailing blanks suppressed	S, M, T, W, T, F, S SUN-SAT Sun-Sat SUNDAYƀƀƀ-SATURDAYƀ Sundayƀƀƀ-Saturdayƀ SUNDAY-SATURDAY Sunday-Saturday	For input, Ws are ignored. For output, W generates uppercase and w generates lowercase. Output padded with blanks (unless Z specified) or truncated to match the number of Ws, up to 20.
All others	delimiters	X'01'-X'FF' (X'00' reserved for run-time environment use)	For input, treated as delimiters between the month, day, year, hour, minute, second, and fraction of a second. For output, copied exactly as is to the target string.

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _FEEDBACK fc;
   _INT4 lil_date1,lil_date2;
   _VSTRING date,date_pic;
   #define SUCCESS "\0\0\0\0"

   /* use CEEDAYS to get the Lilian format */
   strcpy(date.string,"05/14/64");
   date.length = strlen(date.string);
   strcpy(date_pic.string,"MM/DD/YY");
   date_pic.length = strlen(date_pic.string);

   CEEDAYS(&date,&date_pic,&lil_date1,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEDAYS failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }

   /* use CEEDAYS to get the Lilian format */
   strcpy(date.string,"August 14, 1966");

   date.length = strlen(date.string);
   strcpy(date_pic.string,"Mmmmmmmmmmmmz DD, YYYY");
   date_pic.length = strlen(date_pic.string);

   CEEDAYS(&date,&date_pic,&lil_date2,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEDAYS failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }

   /* subtract the two Lilian dates to find out difference in days */
   printf("The number of days between May 14, 1964 and August 14, 1966");
   printf(" is: %d\n",lil_date2 - lil_date1);
}
```

Table 49 (Page 1 of 2). Picture Terms Used in Picture Strings

Picture Terms	Explanation	Valid Values	Notes
Y YY YYY ZYY YYYY	1-digit year 2-digit year 3-digit year 3-digit year within Era 4-digit year	0-9 00-99 000-999 1-999 1582-9999	Y valid for output only. YY assumes range set by CEESCEN. YYY/ZYY used with <JJJJ>, <CCCC> and <CCCCCCCC>.

Examples

1. COBOL/370 Example —

```
01   CHRDATE.
    05 CHRDATE-LENGTH PIC S9(4) COMP.
    05 CHRDATE-STRING PIC X(100).
01   PICSTR.
    05 PICSTR-LENGTH PIC S9(4) COMP.
    05 PICSTR-STRING PIC X(100).
77   LILIAN PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE "Date: 1 January 2000" to CHRDATE-STRING.
     MOVE LENGTH OF CHRDATE-STRING TO CHRDATE-LENGTH.
     MOVE "Date: ZD Mmmmmmmmmmmmmmmz YYYY" TO PICSTR-STRING.
     MOVE LENGTH OF PICSTR-STRING TO PICSTR-LENGTH.
     CALL "CEEDAYS" USING CHRDATE , PICSTR , LILIAN , FC.
```

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EB	3	2507	Insufficient data was passed to CEEDAYS or CEESECS. The Lilian value was not calculated.
CEE2EC	3	2508	The date value passed to CEEDAYS or CEESECS was invalid.
CEE2ED	3	2509	The Japanese or Republic of China Era passed to CEEDAYS or CEESECS was not recognized.
CEE2EH	3	2513	The input date was not within the supported range.
CEE2EL	3	2517	The month value was not recognized.
CEE2EM	3	2518	An invalid picture string was specified.
CEE2EO	3	2520	CEEDAYS detected non-numeric data in a numeric field, or the date string did not match the picture string.
CEE2EP	3	2521	The Japanese or Chinese year-within-Era value passed to CEEDAYS or CEESECS was zero.

Usage Notes

1. The inverse of CEEDAYS is CEEDATE, which converts *output_Lilian_date* from Lilian format to character format.
2. To handle dates earlier than 1582, it is possible to add 4000 to each year, convert to Lilian, calculate, subtract 4000 from the result, and then convert back to character format.
3. By default, 2-digit years are assumed to lie within the 100 year range starting 80 years prior to the system date. Thus, in 1990, all 2-digit years are assumed to represent dates between 1910 and 2009, inclusive. This default range can be changed by using the callable service "CEESCEN — Set the Century Window" on page 387.
4. If *picture_string* includes a Japanese Era symbol <JJJJ>, the YY position in *input_char_date* is replaced by the year number within the Japanese Era. For example, the year 1988 equals the Japanese year 63 in the Showa era. See Table 50 on page 367 for an additional example. See also Table 51 on page 367 for a list of Japanese Eras supported by CEEDATE.
5. If *picture_string* includes an ROC Era symbol <CCCC> or <CCCCCCCC>, the YY position in *input_char_date* is replaced by the year number within the ROC (Republic of China) Era. For example, the year 1988 equals the ROC year 77 in the MinKow Era. See Table 50 on page 367 for an additional example. See Table 52 on page 367 for a list of ROC Eras supported by CEEDATE.
6. Date calculations can be performed easily on the *output_Lilian_date*, since it is an integer. Leap year and end-of-year anomalies are avoided.

input_char_date (input)
a length-prefixed string representing a date or timestamp, in a format that conforms to that specified by *picture_string*.

The character string must contain between 5 and 255 characters, inclusive. *Input_char_date* may contain leading or trailing blanks. Parsing for a date begins with the first non-blank character (unless the picture string itself contains leading blanks, in which case CEEDAYS skips exactly that many positions before parsing begins).

After a valid date is parsed, as determined by the format of the date that you specify in *picture_string*, all remaining characters are ignored by CEEDAYS. Valid dates are in the range between and including the dates 15 October 1582 to 31 December 9999.

See Table 49 on page 365 for a list of valid picture character terms that may be specified in *input_char_date*.

picture_string (input)
a length-prefixed string indicating the format of the date you specified in *input_char_date*.

Each character in the *picture_string* corresponds to a character in *input_char_date*. For example, if you specify MMDDYY as the *picture_string*, CEEDAYS reads an *input_char_date* of 060288 as 02 June 1988.

If delimiters such as the slash (/) appear in the picture string, then leading zeros may be omitted. For example, the following calls to CEEDAYS

```
CALL CEEDAYS('6/2/88'  , 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('06/02/88', 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('060288'  , 'MMDDYY'  , lildate, fc);
CALL CEEDAYS('88154'   , 'YYDDD'   , lildate, fc);
```

would each assign the same value, 148155 (02 June 1988), to *lildate*.

Whenever characters such as colons or slashes are included in the *picture_string* (such as `HH:MI:SS YY/MM/DD`), they count as place-holders but are otherwise ignored.

See Table 49 on page 365 for a list of valid picture characters, and Table 50 on page 367 for examples of valid picture strings.

output_Lilian_date (output)
a 32-bit binary integer representing the Lilian date, the number of days since 14 October 1582. For example, 16 May 1988 is day number 148138.

If *input_char_date* does not contain a valid date, *output_Lilian_date* is set to 0 and CEEDAYS terminates with a non-zero feedback code.

fc (output)
an optional 12-byte feedback code passed by reference indicating the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

callable service ("CEESCEN — Set the Century Window" on page 387) allows you to do this.

For example,

```
Call CEESCEN(30, fc)
```

sets the default century to the 100-year interval starting 30 years prior to the system date, instead of the LE/370 default of 80 years.

A companion service, CEEQCEN ("CEEQCEN — Query the Century Window" on page 385) queries the current century window. This allows, for example, a subroutine to use a different interval for date processing than the parent routine. Before returning, the subroutine would reset the interval back to the way it was previously, as shown in Figure 85 and Figure 86.

```
          CALL "CEEQCEN" USING OLDCEN , QCENFC.
          MOVE 30 TO TEMPCEN.
          CALL "CEESCEN" USING TEMPCEN , SCENFC1.

      *Perform date processing with 2-digit years...

          CALL "CEESCEN" USING OLDCEN , SCENFC2.
```

Figure 85. COBOL/370 example

```
  CEEQCEN ( &OLDCEN , &QCENFC  ;
  TEMPCEN = 30;
  CEESCEN ( &TEMPCEN , &SCENFC1 );

/*Perform date processing with 2-digit years...                    */

  CEESCEN ( &OLDCEN , &SCENFC2 );
```

Figure 86. C/370 example

CEEDAYS — Convert Date to Lilian Format

The CEEDAYS callable service converts a string representing a date into a Lilian format. The Lilian format represents a date as the number of days from the beginning of the Gregorian calendar. Friday, 15 October, 1582 is day one in the Lilian format. The date that you specify in *input_char_date* is therefore converted by CEEDAYS to a number representing the number of days since 14 October, 1582.

This service permits you to perform arithmetic on dates, such as calculating the number of days between two dates.

Syntax

```
▶▶──CEEDAYS──(──input_char_date──,──picture_string──,─────────────▶

▶──output_Lilian_date──,──┬────┬──)───────────────────────────────▶◀
                          └─fc─┘
```

ning of the Gregorian calendar to the current local time) to calculate the employee's total number of days of employment. The final calculation divides that number by 365.25 to get the number of service years.

The range of dates which can be expressed in the Lilian format is 15 October 1582 to 31 December 9999.

Using Calendars Based on Eras

Calendars based on eras use unique strings to identify these eras. These identification strings are called picture strings. The picture strings are used to specify the format of some input and output parameters of date and time callable services. For example, the format of the date 06/16/1990 (where 06 is the month, 16 is the day, and 1990 is the year) corresponds to the picture term MM/DD/YYYY. For additional examples of date formats represented by picture strings, see Table 49 on page 365 and Table 50 on page 367.

An era picture string begins with a less than character < and ends with a greater than character >. The characters inside are either capital J's or capital C's. The following picture strings are supported:

Japanese era is identified by one six-character string <JJJJ>

ROC era has two identification strings. One is a six-character string <CCCC>; the other is a ten-character string <CCCCCCCC>.

National Language Support

National Language Support is provided to date and time services by the NATLANG and COUNTRY run-time options (see "NATLANG" on page 235 and "COUNTRY" on page 224). The names of the months and days of the week are based on the current national language specified in the NATLANG option. In addition, some Date and Time services allow the specification of a blank or null picture string. This directs LE/370 to use a format for date and time based upon the current value specified in the COUNTRY option. The default date/time format for any supported country can be retrieved using the CEEFMDA, CEEFMDT, or CEEFMTM callable services.

Century window routines

To process two-digit years in the year 2000 and beyond, LE/370 employs a sliding scheme whereby all two-digit years are assumed to lie within a one hundred year interval starting 80 years before the current system date:

```
1910                    1990          2009

__________________________|_____________|
                      System Date
```

This is the default for LE/370. For 1990, '10' through '99' is recognized as 1910-1999, and '00' to '09' is recognized as 2000-2009. For 1991, however, '10' is be recognized as 2010 instead of 1910.

By year 2080, all 2-digit years would be recognized as 20xx. In 2081, '00' would be recognized as year 2100.

Some applications may need to set up a different 100-year interval. For example, banks often deal with 30-year bonds, due 01/31/20 for example. The CEESCEN

Chapter 38. Date and Time Services

Introduction

You can use LE/370 date and time services to write, calculate, and read values that represent the date and time. LE/370 offers unique pattern-matching capabilities that permit you to process almost any date and time format contained in an input record or produced by operating system services.

LE/370 includes a complete set of callable services that enhance the capabilities of high level languages to perform date and time calculations. These services can:

- Format date and time values by country code
- Format date and time values using custom formats
- Parse date values and time values
- Convert between Gregorian, Julian, Asian, and Lilian formats
- Calculate days between dates
- Calculate elapsed time to the nearest millisecond
- Obtain local time and GMT from the system without SVC overhead
- Properly handle 2-digit years in the year 2000.

All LE/370 date and time services are enabled for national language support, including full DBCS support for the Japanese Emperor eras and Republic of China era.

Performing Calculations

LE/370 holds dates as a fullword binary integer and timestamps as a doubleword floating point value. This format permits you to perform arithmetic calculations on date and time values in a simple and efficient manner. This eliminates the need to write special subroutines that use services outside of the language library for your application in order to perform these calculations. Figure 84 illustrates an example of how you can use LE/370 callable services to convert a date to a different format and perform a simple calculation on the formatted date.

```
CALL CEEDAYS (date_of_hire, 'YYMMDD', doh_lilian, fc)
CALL CEELOCT (today_Lilian, today_seconds, today_Gregorian, fc)
service_days = today_Lilian - doh_Lilian
service_years = service_days / 365.25
```

Figure 84. Performing Calculations on Dates

In the example, suppose you wanted to calculate the number of years of service for an employee in your organization, and were using the original date of hire in the format YYMMDD to make the calculations. You would use the CEEDAYS service to convert these dates to a Lilian format. The Lilian format, named in honor of the creator of the Gregorian calendar, Luigi Lilio, represents a date as the number of days from the beginning of the Gregorian calendar and is useful for performing calculations involving elapsed time. Day one is Friday, 15 October 1582, and the value is incremented by one for each subsequent day.

In Figure 84, the CEELOCT service is called next to get the current local time. *doh_Lilian* is then subtracted from *today_Lilian* (the number of days from the begin-

© Copyright IBM Corp. 1991

```
  /* print out the default currency symbol for the current country */
  CEE3MCS(country,symbol,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEE3MCS failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("The default currency symbol for %.2s is: ",country);
  printf("%.2s\n\n",symbol);

  /* print out the default thousands separator for the current country */
  CEE3MTS(country,symbol,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEE3MTS failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("The default thousands separator for %.2s is: ",country);
  printf("%.2s\n\n",symbol);
}
```

Figure 83 (Part 2 of 2). C/370 example illustrating calls to CEE3CTY

```
/*************************************************************************/
/* This example shows how to use several of the LE/370 National         */
/* Language Support Callable Services.  The services shown are:         */
/* CEE3CTY  -- determine the current country                            */
/* CEEFMDT  -- get the default date and time format for a country       */
/* CEEFMDS  -- get the default decimal separator for a country          */
/* CEE3MCS  -- get the default currency symbol for a country            */
/* CEE3MTS  -- get the default thousands separator for a country        */
/*************************************************************************/
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

  _FEEDBACK fc;
  _INT4 function;
  _CHAR2 country, symbol;
  _CHAR80 date_pic;
  #define SUCCESS "\0\0\0\0"

  /* query the current country setting */
  function = 2;
  CEE3CTY(&function,country,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEE3CTY failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("The current country is %.2s\n",country);

  /* print out the default date and time format */
  CEEFMDT(country,date_pic,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEFMDT failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("The default date and time setting for %.2s is:\n",country);
  printf("%.80s\n\n",date_pic);

  /* print out the default decimal separator for the current country */
  CEEFMDS(country,symbol,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEFMDS failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  printf("The default decimal separator for %.2s is: ",country);
  printf("%.2s\n\n",symbol);
```

Figure 83 (Part 1 of 2). C/370 example illustrating calls to CEE3CTY

Examples Using Several NLS Services

Figure 82 contains a COBOL/370 example using the CEE3CTY callable service to determine the current country code setting, set the current country code to United States, then later return to the original setting.

Figure 83 on page 357 contains a C/370 example that prints the current country setting, and the default date/time format, decimal separator, currency symbol, and thousands separator for the current country.

```
MOVE 2 TO FUNCTN.
CALL "CEE3CTY" USING FUNCTN , INITCTY , QUERYFC.

MOVE 3 TO FUNCTN.
MOVE "US" TO COUNTRY.
CALL "CEE3CTY" USING FUNCTN , COUNTRY , PUSHFC.

MOVE 4 TO FUNCTN.
CALL "CEE3CTY" USING FUNCTN , COUNTRY , POPFC.

MOVE 2 TO FUNCTN.
CALL "CEE3CTY" USING FUNCTN , COUNTRY , QUERYFC.
```

Figure 82. COBOL/370 example illustrating calls to CEE3CTY

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3C8	2	3464	The thousands separator '*thousands-separator*' was truncated and was not defined in CEE3MTS.
CEE3C9	2	3465	The country code *country-code* was invalid for CEE3MTS. The default thousands separator '*thousands-separator*' was returned.

Usage Notes

1. If you specify an invalid *country_code*, the default thousands separator is ",".
2. To examine the default settings for the country you wish to specify, see Table 59 on page 470.

Examples

1. COBOL/370 Example —

```
77   COUNTRY PIC X(2).
77   THOUSEP PIC X(2).
77   FC PIC X(12).
:

     MOVE SPACE TO COUNTRY.
     CALL "CEE3MTS" USING COUNTRY , THOUSEP , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country,thousand;
   #define SUCCESS "\0\0\0\0"

   /* get the default thousands separator for Canada */
   memcpy(country,"CA",2);
   CEE3MTS(country,thousand,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEE3MTS failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }
   /* print out the default thousands separator */
   printf("The default thousands separator for Canada is: %.2s\n",thousand);
}
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country,currency;
   #define SUCCESS "\0\0\0\0"

   /* get the default currency symbol for Canada */
   memcpy(country,"CA",2);
   CEE3MCS(country,currency,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEE3MCS failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }
   printf("The default currency symbol for Canada is: %.2s\n",currency);
}
```

CEE3MTS —Obtain Default Thousands Separator

The CEE3MTS callable service returns the default thousands separator for the country that you specify in *country_code*.

Syntax

```
▸▸──CEE3MTS──(──country_code──,──thousands_separator──,──────────▸◂

▸──┬────┬──)────────────────────────────────────────────────────▸◂
   └─fc─┘
```

country_code (input)
a 2-character fixed length string representing one of the country codes found in Table 41 on page 225.

country_code is not case sensitive. If no value is specified, the default country code, as set by either the COUNTRY run-time option or the CEE3CTY callable service, is used. See “COUNTRY” on page 224 for an explanation of the COUNTRY run-time option, and “CEE3CTY — Set Default Country” on page 345 for an explanation of the CEE3CTY callable service.

thousands_separator (output)
a 2-character fixed length string representing the default thousands separator for the country specified. The thousands separator is left justified and padded on the right with a blank if necessary.

fc (output)
an optional 12-byte feedback code passed by reference.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument and the requested operation was not successfully completed, the condition is signaled to the condition manager.

currency_symbol (output)
a 4-character fixed length string returned to the calling routine. It contains the default currency symbol for the country specified. The currency symbol is left justified and padded on the right with blanks if necessary.

fc (output)
an optional 12-byte feedback code passed by reference.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3C6	2	3462	The currency symbol '*currency-symbol*' was truncated and was not defined in CEE3MCS.
CEE3C7	2	3463	The country code *country-code* was invalid for CEE3MCS. The default currency symbol '*currency-symbol*' was returned.

Usage Notes

1. If you specify an invalid *country_code*, the default currency symbol is X'9F40'. In the United States, this will be shown as a '$' followed by three blanks.
2. To examine the default settings for the country you wish to specify, see Table 59 on page 470.

Examples

1. COBOL/370 Example —

```
77   COUNTRY PIC X(2).
77   CURSYM PIC X(2).
77   FC PIC X(12).
:

     MOVE SPACE TO COUNTRY.
     CALL "CEE3MCS" USING COUNTRY , CURSYM , FC.
```

Table 48 (Page 2 of 2). National Language Codes

ID	National Language	Original Name	Principal Country
ELL	Greek	Ellinika	Greece
ENG	UK English	English	United Kingdom
ENU	US English		United States
ESP	Spanish	Espanol	Spain
FIN	Finnish	Suomi	Finland
FRA	French	Francais	France
FRB	Belgian French		Belgium
FRC	Canadian French		Canada
FRS	Swiss French	Suisse-francais	Switzerland
HEB	Hebrew	Ivrith	Israel
HUN	Hungarian	Magyar	Hungary
ISL	Icelandic	Islenska	Iceland
ITA	Italian	Italiano	Italy
ITS	Swiss Italian	Italiano svizzero	Switzerland
JPN	Japanese	Nihongo	Japan
KOR	Korean	Choson-o; Hanguk-o	Korea
NLD	Dutch	Nederlands	Netherlands
NLB	Belgian Dutch		Belgium
NOR	Norwegian - Bokmal	Norsk - Bokmal	Norway
NON	Norwegian - Nynorsk	Norsk - Nynorsk	Norway
PLK	Polish	Polski	Poland
PTG	Portuguese	Portugues	Portugal
PTB	Brazilian Portuguese		Brazil
RMS	Rhaeto-Romanic	Romontsch	Switzerland
ROM	Romanian	Romana	Romania
RUS	Russian	Russkij	USSR
SHC	Serbo-Croatian (Cyr)	Srpsko-hrvatski	Yugoslavia
SHL	Serbo-Croatian (Lat)		Yugoslavia
SKY	Slovakian	Slovensky	Czechoslovakia
SQI	Albanian	Shqip	Albania
SVE	Swedish	Svenska	Sweden
THA	Thai	Thai	Thailand
TRK	Turkish	Turkce	Turkey
UEN	US Uppercase English		United States
URD	Urdu	Urdu	Pakistan

CEE3MCS — Obtain Default Currency Symbol

The CEE3MCS callable service returns the default currency symbol for the country you specify in *country_code*.

Syntax

```
▸▸──CEE3MCS──(──country_code──,──currency_symbol──,──┬────┬──)──▸◂
                                                     └─fc─┘
```

country_code (input)

a 2-character fixed length string representing one of the country codes found in Table 41 on page 225.

country_code is not case sensitive. If no value is specified, the default country code, as set by either the COUNTRY run-time option or the CEE3CTY callable service, is used. See "COUNTRY" on page 224 for an explanation of the COUNTRY run-time option, and "CEE3CTY — Set Default Country" on page 345 for an explanation of the CEE3CTY callable service.

```
77   FUNCTN PIC S9(9) COMP.
77   LANG PIC X(3).
77   FC PIC X(12).
:

     MOVE 2 TO FUNCTN.
     CALL "CEE3LNG" USING FUNCTN , LANG , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

  _FEEDBACK fc;
  _INT4 function;
  _CHAR3 lang;
  #define SUCCESS "\0\0\0\0"

  /* Query the current language setting */
  function = 2;
  CEE3LNG(&function,lang,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEE3LNG failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }

  /* if the current language is not mixed-case American English */
  /* set the current language to mixed-case American English    */
  if (memcmp(lang,"ENU",3) != 0) {
     memcpy(lang,"ENU",3);
     function = 1;
     CEE3LNG(&function,lang,&fc);
     if (memcmp(&fc,SUCCESS,4) != 0) {
        printf("CEE3LNG failed with message number %d\n",fc.tok_msgno);
        exit(2999);
     }
  }
}
```

Table 48 (Page 1 of 2). National Language Codes

ID	National Language	Original Name	Principal Country
AFR	Afrikaans	Afrikaans	South Africa
ARA	Arabic	Arabi	Arab Countries
BGR	Bulgarian	Bulgarski	Bulgaria
CAT	Catalan	Catala	Spain
CHT	Traditional Chinese	Zhongwen	R.O.C.
CHS	Simplified Chinese		P.R.C.
CSY	Czech	Cesky	Czechoslovakia
DAN	Danish	Dansk	Denmark
DEU	German	Deutsch	Germany
DES	Swiss German	Schweizer-Deutsch	Switzerland

ation was not successfully completed, the condition is signaled to the condition manager. The following symbolic conditions may result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3BQ	2	3450	Only one language was on the stack when a POP request was made to CEE3LNG. The current language was returned in the desired language parameter.
CEE3BR	3	3451	The desired language *desired-language* for the PUSH or SET function for CEE3LNG was invalid. No operation was performed.
CEE3BS	3	3452	The function *function* specified for CEE3LNG was not recognized. No operation was performed.

Usage Notes

1. More than one national language can be PUSHed. Languages are POPped in a last-in first-out (LIFO) order.
2. Message modules associated with a national language are not loaded until they are actually needed in order to format a message.
3. If you specify a *national_language* that is unavailable on your system, the IBM-supplied default *national_language* is used. In LE/370, this is ENU (mixed case U.S. English). CEEUOPT and CEEDOPT permit the specification of an unknown national language code, but give a return code of 4 and a warning message. If an invalid language is specified, the IBM-supplied default ENU (mixed case U.S. English) is used. See Chapter 30, "Specifying Run-time Options" on page 206 for more information.
4. C/370 Considerations — C/370 provides locales that map to German, Spanish, French, US English, and others. They establish default formats for such things as currency symbols, and names of the days of the week. To change the locale, you can use the `setlocale()` library function.

   The settings of `setlocale()` and the CEE3LNG run-time option *do not* affect one another. CEE3LNG affects only LE/370 services and error messages; `setlocale()` affects only C/370 functions.

   To ensure that everything is set properly for your country, use both CEE3LNG and `setlocale()`. For more information, see the *IBM SAA AD/Cycle C/370 Programming Guide.*

Examples

1. COBOL/370 Example —

1	SET — establishes the *desired_language* specified in the call to CEE3LNG as the current language. It effectively replaces the current language on the top of the stack with the *desired_language* that you specify. When setting the national language, the *desired_language* is folded to uppercase. "enu" and "ENU", for example, are considered to be the same national language.
2	QUERY — identifies the current language on the top of the stack to the calling routine by returning it in the *desired_language* parameter of CEE3LNG. The *desired_language* retained as the result of the QUERY function is in uppercase.
3	PUSHes the *desired_language* specified in the call to CEE3LNG onto the top of the language stack, making it the current language. Previous languages are retained on the stack on a LIFO basis. This makes it possible to return to a prior language at a later time.
4	POPs the current language off the stack. The previous language that was PUSHed onto the stack now becomes the new current language. Upon return to the caller, the *desired_language* parameter contains the discarded language. If the stack contains only one language and would be empty after the call, no action is taken and a feedback code indicating such is returned.

desired_language (input/output)
a 3-character fixed length string. The string is not case-sensitive and is used in the following ways for different *functions:*

If *function* is specified as:	Then *desired_language*:
1 or 3	Contains the desired national language identification. In this case, it is an input parameter. Table 48 on page 351 contains a list of national language identifiers. **Note:** LE/370 supports only these national languages in Release 1: ENU Mixed-case American English UEN Uppercase American English JPN Japanese.
2	Returns the current language on top of the stack. In this case, it is an output parameter.
4	Returns the discarded national language. In this case, it is an output parameter.

fc (output)
an optional 12-byte feedback code passed by reference.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified and the requested oper-

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

  _FEEDBACK fc;
  _INT4 function;
  _CHAR2 country;
  #define SUCCESS "\0\0\0\0"

  /* query the current country setting */
  function = 2;
  CEE3CTY(&function,country,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEE3CTY failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }

  /* if the current country is not Canada then set it to Canada */
  if (memcmp(country,"CA",2) != 0) {
     memcpy(country,"CA",2);
     function = 1;
     CEE3CTY(&function,country,&fc);

     if (memcmp(&fc,SUCCESS,4) != 0) {
        printf("CEE3CTY failed with message number %d\n",fc.tok_msgno);
        exit(2999);
     }
  }
}
```

CEE3LNG — Set National Language

The CEE3LNG callable service allows the calling routine to change or query the current national language. The national languages can be recorded on a last-in first-out (LIFO) national language stack.

Changing the national language changes the languages in which error messages are displayed and printed, the names of the days of the week, and the names of the months.

Syntax

```
▸▸──CEE3LNG──(──function──,──desired_language──,──┬────┬──)──────▸◂
                                                  └─fc─┘
```

function (input)

a fullword binary integer.

function specifies the service that should be performed. *function* is not case-sensitive. The possible values for *function* are:

Usage Notes

1. To examine the default settings for the country you wish to specify, see Table 59 on page 470.
2. If you need to override your default country setting, you may do so with the CEE3CTY callable service. For example, if you live in the United States, you would have specified the code for the United States as the default at installation. If in a certain application you wanted to use the French defaults, however, you could use the CEE3CTY service to SET France as the national country setting. Then when you again wanted the defaults for the United States, you would PUSH the code for United States back to the top of the stack.
3. The bytes X'0E' and X'0F' representing shift-out and shift-in codes are not affected by any *country_code* setting.
4. The current national country setting affects the format of the date and time in the reports generated by the RPTOPTS and RPTSTG run-time options. For more information on these run-time options, see "RPTOPTS" on page 239 and "RPTSTG" on page 240.
5. CEE3CTY also affects the default symbols for the National Language Support and date/time callable services.
6. C/370 Considerations — C/370 provides locales that map to German, Spanish, French, US English, and others. They establish default formats for such things as currency symbols, and names of the days of the week. To change the locale, you can use the `setlocale()` library function.

   The settings of `setlocale()` and the CEE3CTY run-time option *do not* affect one another. CEE3CTY affects only LE/370 services; `setlocale()` affects only C/370 functions.

   To ensure that everything is set properly for your country, use both CEE3CTY and `setlocale()`. For more information, see the *IBM SAA AD/Cycle C/370 Programming Guide.*

Examples

1. COBOL/370 Example —

```
77   FUNCTN PIC S9(9) COMP.
77   COUNTRY PIC X(2).
77   FC PIC X(12).
:

     MOVE 2 TO FUNCTN.
     CALL "CEE3CTY" USING FUNCTN , COUNTRY , FC.
```

4	POPs the current country code. The last country code that was PUSHed now becomes the current *country_code*. Upon return to the calling routine, the *country_code* parameter contains the discarded country code. If the stack contains only one country code, the code cannot be POPped because the stack would be empty after the call. Therefore, no action is taken and a feedback code indicating such is returned to the calling routine.

country_code(input/output)
a 2-character fixed length string

country_code is not case-sensitive. It is used in the following ways for the different *functions:*

If *function* **is specified as:**	**Then** *country_code*:
1 or 3	Contains the desired 2-character country code. In this case, it is an input parameter. Table 41 on page 225 contains a list of valid country codes.
2	Returns the current 2-character country code on top of the stack. In this case, it is an output parameter.
4	Returns the discarded 2-character country code. In this case, it is an output parameter.

fc (output)
an optional 12-byte feedback code passed by reference.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager.

The following symbolic conditions may result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3BV	2	3455	Only one country code was on the stack when a POP request was made to CEE3CTY. The current country code was returned in the country code parameter.
CEE3C0	3	3456	The country code *country-code* for the PUSH or SET function for CEE3CTY was invalid. No operation was performed.
CEE3C1	3	3457	The function *function* specified for CEE3CTY was not recognized. No operation was performed.

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country;
   _CHAR80 time_pic;
   #define SUCCESS "\0\0\0\0"

   /* get the default time format for Canada */
   memcpy(country,"CA",2);
   CEEFMTM(country,time_pic,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEFMTM failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }
   /* print out the default time format */
   printf("%.80s\n",time_pic);
}
```

CEE3CTY — Set Default Country

The CEE3CTY callable service allows the calling routine to change or query the current national country setting. The country setting affects the date format, the time format, the currency symbol, the decimal separator character, and the thousands separator.

Syntax

```
►►──CEE3CTY──(──function──,──country_code──,──┬────┬──)──────────►◄
                                              └─fc─┘
```

function (input)

a fullword binary integer that specifies the service to be performed.

function is not case-sensitive. The possible values for *function* are:

1	SET establishes the *country_code* parameter as the current country. The top of the stack is therefore effectively replaced with the *country_code*.
2	QUERY returns the current country code on the top of the stack to the calling routine. The current code is returned in the *country_code* parameter.
3	PUSHes the *country_code* parameter onto the top of the country code stack, making it the current country code. Previous country codes on the stack are retained on a LIFO basis. This makes it possible to return to a prior country code at a later time.

ation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3CD	2	3469	The country code *country-code* was invalid for CEEFMTM. The default time picture string *time-pic-string* was returned.
CEE3CE	2	3470	The date and time string *datetime-string* was truncated and was not defined in CEEFMTM.

Usage Notes

1. If you specify an invalid *country_code*, the default time picture string is "HH:MI:SS".
2. To examine the default settings for the country you wish to specify, see Table 59 on page 470.

Examples

1. COBOL/370 Example —

```
77   COUNTRY PIC X(2).
77   PICSTR PIC X(80).
77   FC PIC X(12).
 :

     MOVE SPACE TO COUNTRY.
     CALL "CEEFMTM" USING COUNTRY , PICSTR , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country;
   _CHAR80 date_pic;
   #define SUCCESS "\0\0\0\0"

   /* get the default date and time format for Canada */
   memcpy(country,"CA",2);
   CEEFMDT(country,date_pic,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEFMDT failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }
   /* print out the default date and time format */
   printf("%.80s\n",date_pic);
}
```

CEEFMTM — Obtain Default Time Format

The CEEFMTM callable service returns to the calling routine the default time picture string for the country specified in the *country_code*.

Syntax

```
▸▸──CEEFMTM──(──country_code──,──time_pic_str──,──┬────┬──)──────▸◂
                                                  └─fc─┘
```

country_code (input)
a 2-character fixed length string representing one of the country codes found in Table 41 on page 225.

country_code is not case sensitive. If no value is specified, the default country code, as set by either the COUNTRY run-time option or the CEE3CTY callable service, is used. See "COUNTRY" on page 224 for an explanation of the COUNTRY run-time option, and "CEE3CTY — Set Default Country" on page 345 for an explanation of the CEE3CTY callable service.

time_pic_str (output)
an 80-character fixed length string representing the default time picture string for the country specified. The picture string is left justified and padded on the right with blanks if necessary.

fc (output)
an optional 12-byte feedback code passed by reference.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested oper-

ation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3CF	2	3471	The country code *country-code* was invalid for CEEFMDT. The default date and time picture string *datetime-string* was returned.

Usage Notes

1. If you specify an invalid *country_code*, the default date/time picture string is "YYYY-MM-DD HH:MI:SS".
2. To examine the default settings for the country you wish to specify, see Table 59 on page 470.

Examples

1. COBOL/370 Example —

```
77   COUNTRY PIC X(2).
77   PICSTR PIC X(80).
77   FC PIC X(12).
:

     MOVE SPACE TO COUNTRY.
     CALL "CEEFMDT" USING COUNTRY , PICSTR , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country,decimal;
   #define SUCCESS "\0\0\0\0"

   /* get the default decimal separator for Canada */
   memcpy(country,"CA",2);
   CEEFMDS(country,decimal,&fc);
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEFMDS failed with message number %d\n",fc.tok_msgno);
      exit(2999);
   }
   /* print out the default decimal separator */
   printf("The default decimal separator for Canada is: %.2s\n",decimal);
}
```

CEEFMDT — Obtain Default Date and Time Format

The CEEFMDT callable service returns the default date and time picture strings for the country specified in the *country_code*.

Syntax

```
▸▸——CEEFMDT—(—country_code—,—datetime_str——,—┬————┬—)————————▸◂
                                             └-fc-┘
```

country_code (input)
a 2-character fixed length string representing one of the country codes found in Table 41 on page 225.

country_code is not case sensitive. If no value is specified, the default country code, as set by either the COUNTRY run-time option or the CEE3CTY callable service, is used. See "COUNTRY" on page 224 for an explanation of the COUNTRY run-time option, and "CEE3CTY — Set Default Country" on page 345 for an explanation of the CEE3CTY callable service.

datetime_str (output)
an 80-character fixed length string returned to the calling routine. It contains the default date and time picture string for the country specified. The picture string is left justified and padded on the right with blanks, if necessary.

fc (output)
an optional 12-byte feedback code passed by reference.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested oper-

COUNTRY run-time option, and “CEE3CTY — Set Default Country” on page 345 for an explanation of the CEE3CTY callable service.

OUTPUT ???

decimal_separator (input)

a 2-character fixed length string containing the default decimal separator for the country specified. The decimal separator is left justified and padded on the right with a blank if necessary.

fc (output)

an optional 12-byte feedback code passed by reference.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3C4	2	3460	The decimal separator '*decimal-separator*' was truncated and was not defined in CEE3MDS.
CEE3C5	2	3461	The country code *country-code* was invalid for CEE3MDS. The default decimal separator '*decimal-separator*' was returned.

Usage Notes

1. If you specify an invalid *country_code*, the default decimal separator is “.”.
2. To examine the default settings for the country you wish to specify, see Table 59 on page 470.

Examples

1. COBOL/370 Example —

```
77   COUNTRY PIC X(2).
77   DECSEP PIC X(2).
77   FC PIC X(12).
:

     MOVE SPACE TO COUNTRY.
     CALL "CEEFMDS" USING COUNTRY , DECSEP , FC.
```

Examples

1. COBOL/370 Example —

```
77   COUNTRY PIC X(2).
77   PICSTR PIC X(80).
77   FC PIC X(12).
:

     MOVE SPACE TO COUNTRY.
     CALL "CEEFMDA" USING COUNTRY , PICSTR , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

  _FEEDBACK fc;
  _CHAR2 country;
  _CHAR80 date_pic;
  #define SUCCESS "\0\0\0\0"

  /* get the default date format for Canada */
  memcpy(country,"CA",2);
  CEEFMDA(country,date_pic,&fc);
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEFMDA failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  /* print out the default date format for Canada */
  printf("%.80s\n",date_pic);
}
```

LEAWI.H incorrect

should this be "&date_pic" It is an output parameter

CEEFMDS — Obtain Default Decimal Separator

The CEEFMDS callable service returns the default decimal separator for the country specified in the *country_code*.

Syntax

```
▸▸——CEEFMDS—(—country_code—,—decimal_separator—,—┬────┬—)—▸◂
                                                └─fc─┘
```

country_code (input)

a 2-character fixed length string representing one of the country codes found in Table 41 on page 225.

country_code is not case-sensitive. If no value is specified, the default country code, as set by either the COUNTRY run-time option or the CEE3CTY callable service, is used. See "COUNTRY" on page 224 for an explanation of the

Chapter 37. National Language Support

CEEFMDA — Obtain Default Date Format

The callable service CEEFMDA returns to the calling routine the default date picture string for a specified country.

```
Syntax
►►──CEEFMDA──(──country_code──,──date_pic_str──,──┬────┬──)──────────►◄
                                                   └─fc─┘
```

country_code (input)
a 2-character fixed length string representing one of the country codes found in Table 41 on page 225.

country_code is not case-sensitive. If no value is specified, the default country code, as set by either the COUNTRY run-time option or the CEE3CTY callable service, is used. See "COUNTRY" on page 224 for an explanation of the COUNTRY run-time option, and "CEE3CTY — Set Default Country" on page 345 for an explanation of the CEE3CTY callable service.

date_pic_str (output)
an 80-character fixed length picture string returned to the calling routine. It contains the default date picture string for the country specified. The picture string is left justified and padded on the right with blanks if necessary.

fc (output)
an optional 12-byte feedback code passed by reference.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3CB	2	3467	The country code *country-code* was invalid for CEEFMDA. The default date picture string *date-pic-string* was returned.

Usage Notes

1. If you specify an invalid *country_code*, the default date format is "YYYY-MM-DD".
2. To examine the default settings for the country you wish to specify, see Table 59 on page 470.

© Copyright IBM Corp. 1991

```
  /* use CEEMGET until all the message has been retrieved        */
  /* msgindx will be zero when all the message has been retrieved */
  do {
     CEEMGET(&token,msgarea,&msgindx,&fc);

     if (fc.tok_sev > 1) {
        dest = 2;
        CEEMSG(&fc,&dest,NULL);
        exit(2999);
     }
     memcpy(message.string,msgarea,80);
     message.length = 80;
     dest = 2;

     CEEMOUT(&message,&dest,&fc);           /* put out the message */

     if (memcmp(&fc,SUCCESS,4) != 0) {
        dest = 2;
        CEEMSG(&fc,&dest,NULL);
        exit(2999);
     }
  } while (msgindx != 0);
}
```

Figure 81 (Part 2 of 2). C/370 example illustrating calls to CEEMGET, CEENCOD, and CEEMSG callable services

```
/***********************************************************************/
/* This example shows how all the LE/370 Message Handling Callable    */
/* Services are used.  The services shown are:                        */
/* CEEMOUT  --  dispatch a message                                    */
/* CEEMSG   --  retrieve, format and output a message                 */
/* CEEMGET  --  retrieve a message                                    */
/*                                                                    */
/* The example also shows how to directly construct a condition token.*/
/***********************************************************************/
#include <leawi.h>
#include <string.h>
#include <stdio.h>

#define SUCCESS "\0\0\0\0"

int main(void) {

  _VSTRING message;
  _INT4 dest,msgindx;
  _CHAR80 msgarea;
  _FEEDBACK fc,token;

  strcpy(message.string,"This is a test message");
  message.length = strlen(message.string);
  dest = 2;

  CEEMOUT(&message,&dest,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     /* put the message if CEEMOUT failed */
     dest = 2;
     CEEMSG(&fc,&dest,NULL);
     exit(2999);
  }

  /* construct a token for CEE message 0260 */
  token.tok_sev = 3;
  token.tok_msgno = 260;
  token.tok_case = 1;
  token.tok_sever = 3;
  token.tok_ctrl = 1;
  memcpy(token.tok_facid,"CEE",3);
  token.tok_isi = 0;

  /* initialize the message area */
  msgindx = 0;
  memset(msgarea,' ',79);
  msgarea[80] = '\0';
```

Figure 81 (Part 1 of 2). C/370 example illustrating calls to CEEMGET, CEENCOD, and CEEMSG callable services

```
*Call CEEMSG to output any other message from CEEMGET
         IF ( MSGNO NOT = 0 )
           THEN
             CALL "CEEMSG" USING MGETFC , MSGDEST , MSGFC;
         END-IF;
       ELSE
*
*Call CEEMSG to output any message from CEENCOD
         CALL "CEEMSG" USING NCODFC , MSGDEST , MSGFC;
     END-IF.
```

Figure 80 (Part 2 of 2). COBOL/370 example illustrating calls to CEEMGET, CEENCOD, and CEEMSG services

```
*Call CEEMOUT to put out header message
     MOVE 2 TO MSGDEST.
     MOVE "Test of message services" TO MSGSTR-STRING.
     MOVE 80 TO MSGSTR-LENGTH.
     CALL "CEEMOUT" USING MSGSTR , MSGDEST , MOUTFC.
*
*Set up CEENCOD call, we want msg 3456 for facility id CEE
*This message number is for no active condition
     MOVE 3 TO SEV.
     MOVE 260 TO MSGNO.
     MOVE 1 TO CASE.
     MOVE 3 TO SEV2.
     MOVE 1 TO CNTRL.
     MOVE "CEE" TO FACID.
     MOVE 0 TO ISINFO.
*
*Call CEENCOD to construct ctok from input parameters
     CALL "CEENCOD" USING SEV , MSGNO , CASE , SEV2 , CNTRL ,
     FACID , ISINFO , CTOK , NCODFC.
*
*If call to CEENCOD was successful then
     IF ( NCODFC(1:8) = LOW-VALUE )
       THEN
*
*Set up call to CEEMGET
         MOVE 0 TO MSGINDX;
         MOVE SPACE TO MSGAREA;
*
*Want to loop until entire message has been retrieved
         PERFORM TEST AFTER UNTIL ( MSGNO NOT = 455 )
*
*Call CEEMGET to retrieve each portion of the message
           CALL "CEEMGET" USING CTOK , MSGAREA , MSGINDX , MGETFC;
*
*Call CEEDCOD to decode feedback code from CEEMGET
           CALL "CEEDCOD" USING MGETFC , SEV , MSGNO , CASE ,
           SEV2 , CNTRL , FACID , ISINFO , DCODFC;
*
*Call CEEMOUT to output portion of message retrieved
           MOVE MSGAREA TO MSGSTR-STRING;
           CALL "CEEMOUT" USING MSGSTR , MSGDEST , MOUTFC;
         END-PERFORM;
*
```

Figure 80 (Part 1 of 2). COBOL/370 example illustrating calls to CEEMGET, CEENCOD, and CEEMSG services

Examples

1. COBOL/370 Example —

```
77   CONTOK PIC X(12).
77   MSGDEST PIC S9(9) COMP.
77   FC PIC X(12).
 :

     MOVE 2 TO MSGDEST.
     CALL "CEEMSG" USING CONTOK , MSGDEST , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

#define SUCCESS "\0\0\0\0"

int main(void) {

  _VSTRING message;
  _INT4 dest,msgindx;
  _CHAR80 msgarea;
  _FEEDBACK fc,token;

  strcpy(message.string,"This is a test message");
  message.length = strlen(message.string);
  dest = 2;

  CEEMOUT(&message,&dest,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     /* put the message if CEEMOUT failed */
     dest = 2;
     CEEMSG(&fc,&dest,NULL);
     exit(2999);
  }
}
```

Examples Using Several Message Handling Services

Figure 80 on page 334 contains a COBOL/370 routine illustrating calls to LE/370 message handling callable services to obtain a message associated with a given condition token.

Figure 81 on page 336 illustrates a C/370 routine which calls all of the LE/370 message handling services.

Syntax

```
▸▸——CEEMSG—(—Cond_Token—,—Destination_Code—,——————)——————▸◂
                                            └fc┘
```

Cond_Token (input)
a 12-byte condition token received as the result of an LE/370 callable service.

Destination_Code (input)
a 4-byte binary integer.

Destination_Code can only be specified as 2, meaning write the message to the *ddname* of the file specified in the MSGFILE run-time option. See "MSGFILE" on page 234 for more information.

fc (output)
a 12-byte feedback code passed by reference indicating the result of the service.

The feedback information in the form of a condition token is returned to the calling routine. The following conditions may result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0E2	3	0450	The message inserts for the condition token with message number *message-number* and facility ID *facility-id* could not be located.
CEE0E3	3	0451	An unsupported destination code *destination-code* was passed to routine *routine-name.*
CEE0E6	3	0454	The message number *message-number* could not be found for facility ID *facility-id.*
CEE0E9	3	0457	The MSGFILE destination *ddname* could not be located.
CEE0EA	3	0458	The message repository *repository-name* could not be located.
CEE3CT	3	3485	An internal message services error occurred while locating the message number within the message file.
CEE3CU	3	3486	An internal message services error occurred while formatting the message.
CEE3CV	3	3487	An internal message services error occurred while locating the message number within the ranges specified in the repository.

Examples

1. COBOL/370 Example —

```
01   MSGSTR.
    05 MSGSTR-LENGTH PIC S9(4) COMP.
    05 MSGSTR-STRING PIC X(80).
77   DESTIN PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE "CEEMOUT test" TO MSGSTR-STRING.
     MOVE 12 TO MSGSTR-LENGTH.
     MOVE 2 TO DESTIN.
     CALL "CEEMOUT" USING MSGSTR , DESTIN , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

#define SUCCESS "\0\0\0\0"

int main(void) {

  _VSTRING message;
  _INT4 dest;
  _FEEDBACK fc;

  strcpy(message.string,"This is a test message");
  message.length = strlen(message.string);
  dest = 2;

  CEEMOUT(&message,&dest,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEMOUT failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  .
  .
  .
}
```

CEEMSG — Get, Format, and Dispatch a Message

The CEEMSG service is designed to obtain/format/dispatch a message corresponding to an input condition token received from a callable service or passed to a user-written condition handler. You can use this service to print a message after a call to any LE/370 service which returns a condition token.

CEEMOUT — Dispatch a Message

The CEEMOUT callable service dispatches a user-defined message to the message file.

Syntax

```
▸▸——CEEMOUT—(—Message_String—,—Destination_Code—,——————————▸

▸——┬———┬—)———————————————————————————————————————————————▸◂
   └fc—┘
```

Message_String (input)

a halfword prefixed string containing the message. DBCS characters must be enclosed within shift-out shift-in characters (that is, bytes X'0F' and X'0E', respectively).

Insert data cannot be placed in the message with this call.

Destination_Code (input)

a 4-byte binary integer.

The *only* accepted value for *Destination_Code* is 2. Under systems other than CICS, LE/370 then writes the message to the *ddname* of the file specified in the MSGFILE run-time option. Under CICS, the message is written to a transient data queue named CESE. See "MSGFILE" on page 234 for more information.

fc (output)

an optional 12-byte feedback code passed by reference indicating the result of the service.

Feedback information in the form of a condition token is returned to the calling routine. The following conditions can result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0E3	3	0451	An unsupported destination code *destination-code* was passed to routine *routine-name*.
CEE0E9	3	0457	The MSGFILE destination *ddname* could not be located.

2. C/370 Example —

```
#include <leawi.h>
#include <string.h>
#include <stdio.h>

#define SUCCESS "\0\0\0\0"

int main(void) {

  _VSTRING message;
  _INT4 dest,msgindx;
  _CHAR80 msgarea;
  _FEEDBACK fc,token;

  /* construct a token for CEE message 0260 */
  token.tok_sev = 3;
  token.tok_msgno = 260;
  token.tok_case = 1;
  token.tok_sever = 3;
  token.tok_ctrl = 1;
  memcpy(token.tok_facid,"CEE",3);
  token.tok_isi = 0;

  msgindx = 0;
  memset(msgarea,' ',79);     /* initialize the message area */
  msgarea[80] = '\0';

  /* use CEEMGET until all the message has been retrieved        */
  /* msgindx will be zero when all the message has been retrieved */
  do {
     CEEMGET(&token,msgarea,&msgindx,&fc);

     if (fc.tok_sev > 1 ) {
        printf("CEEMGET failed with message number %d\n",fc.tok_msgno);
        exit(2999);
     }

     /* put out the message using CEEMOUT */
     memcpy(message.string,msgarea,80);
     message.length = 80;
     dest = 2;
     CEEMOUT(&message,&dest,&fc);

     if (memcmp(&fc,SUCCESS,4) != 0) {
        printf("CEEMOUT failed with message number %d\n",fc.tok_msgno);
        exit(2999);
     }
  } while (msgindx != 0);
}
```

routine. The following conditions may result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE036	3	0102	An unrecognized condition token was passed to *routine* and could not be used.
CEE0E2	3	0450	The message inserts for the condition token with message number *message-number* and facility ID *facility-id* could not be located.
CEE0E6	3	0454	The message number *message-number* could not be found for facility ID *facility-id*.
CEE0E7	1	0455	The message with message number *message-number* and facility ID *facility-id* was truncated.
CEE0EA	3	0458	The message repository *repository-name* could not be located.

Examples

1. COBOL/370 Example —

```
77   CONTOK PIC X(12).
77   MSGBUF PIC X(80).
77   MSGINDEX PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE 0 TO MSGINDEX.
     CALL "CEEMGET" USING CONTOK , MSGBUF , MSGINDEX , FC.
```

Chapter 36. Message Handling

CEEMGET — Get a Message

The CEEMGET service retrieves, formats and stores in a buffer a message corresponding to a condition token either returned for a callable service or passed to a user-written condition handler. The message may later be retrieved for manipulation or output by the caller.

Syntax

```
▸▸——CEEMGET——(——Cond_Token—,—Message_Area—,——Msg_Index————▸

▸————┬──┬——)——————————————————————————————————————————————▸◂
     └fc┘
```

Cond_Token (input)
a 12-byte condition token received as the result of an LE/370 callable service. See "CEENCOD — Construct a Condition Token" on page 306 for a description of this condition token.

Message_Area (input/output)
a fixed length 80-character string where the message is placed.

The message is left justified and padded right with blanks.

Msg_Index (input/output)
a 4-byte binary integer that is returned to the calling routine.

The *Msg_Index* should be passed a value of zero on the initial call to CEEMGET. If a message is too large to be contained in the *Message_Area*, *Msg_Index* is returned containing the index into the message. This index is used on subsequent calls to CEEMGET to retrieve the remaining portion of the message. A feedback code is also returned to indicate that the message has been truncated. When the entire message has been returned, *Msg_Index* is zero.

fc (output)
a 12-byte feedback code passed by reference indicating the result of the service.

Feedback information in the form of a condition token is returned to the calling

```
    /* get the q_data_token */
    CEEGQDT(fc,&qdata,NULL);

    /* look at the q_data_token and print out a message if the */
    /* error_value was 86                                      */
    if (((info_struct *)qdata)->error_value == 86)
       printf("%.80s\n",((info_struct*)qdata)->err_msg);

    /* move the resume cursor to the caller of the routine */
    /* that registered this handler                        */
    type = 1;
    CEEMRCR(&type,&cursorfc);

    if (memcmp(&cursorfc,SUCCESS,4) != 0) {
       printf("CEEMRCR failed with message number %d\n",cursorfc.tok_msgno);
       exit (2999);
    }

    /* mark the condition as handled and return */
    printf("Condition handled\n");
    *result = 10;
    return;
 }
```

Figure 79 (Part 3 of 3). Sample C/370 calls to several LE/370 condition management callable services

```
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEHDLR failed with message number %d\n",fc.tok_msgno);
     exit (2999);
  }

  /* build the condition token */
  condtok.tok_sev = 1;
  condtok.tok_msgno = 99;
  condtok.tok_case = 1;
  condtok.tok_sever = 1;
  condtok.tok_ctrl = 0;
  memcpy(condtok.tok_facid,"ZZZ",3);
  condtok.tok_isi = 0;

  /* set up the condition info structure */
  info = malloc(sizeof(info_struct));
  if (info == NULL) {
     printf("error allocating info_struct\n");
     exit(2399);
  }
  memset(info->err_msg,' ',79);
  info->err_msg[80] = '\0';
  info->error_value = 86;
  memcpy(info->err_msg,"Test message",12);
  info->retcode = 99;

  /* set qdata to be the condition info structure */
  qdata = (int)info;
  /* signal the condition */
  CEESGL(&condtok,&qdata,NULL);

  printf("Failed: handler should have moved resume cursor past this\n");
}

/*******************************************************************/
/* User condition handler                                          */
/*******************************************************************/
void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {

   _FEEDBACK cursorfc, orig_fc;
   _INT4 type;
   _INT4 qdata;


   /* if the condition is not mine (ZZZ facid) then percolate */
   if (memcmp(fc->tok_facid,"ZZZ",3) != 0) {
      *result = 20;
      return;
   }

   printf("%d is handling the condition for Control\n",*token);
```

Figure 79 (Part 2 of 3). Sample C/370 calls to several LE/370 condition management callable services

```
/**************************************************************************/
/* This example shows how several of the LE/370 Condition Management   */
/* Callable Services are used.  The services shown are:                 */
/* CEEHDLR  -- register a user condition handler                        */
/* CEESGL   -- signal a condition to the condition manager              */
/* CEEGQDT  -- get the q_data_token                                     */
/* CEEMRCR  -- move the resume cursor                                   */
/*                                                                      */
/* The example also shows how to directly construct a condition token */
/* and provides a sample user condition handler.                        */
/**************************************************************************/
#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void b(void);
void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#define SUCCESS "\0\0\0\0"

typedef struct {          /* condition info structure */
   int   error_value;
   char err_msg[80];
   int   retcode;
} info_struct;

int main(void) {

  printf("In main program\n");
   b();
  /* CEEMRCR should put the resume cursor at this point */
  printf("Finished\n");
}
void b(void) {

  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;
  info_struct *info;

  printf("In routine b\n");
  token = 99;
  routine.address = (_POINTER)&handler;
  routine.nesting = NULL;
  /* register the condition handler: handler */
  CEEHDLR(&routine,&token,&fc);
```
Figure 79 (Part 1 of 3). Sample C/370 calls to several LE/370 condition management callable services

```
  if (memcmp(&fc,SUCCESS,4) != 0) {
    printf("CEE3SPM query failed with message %\n",fc.tok_msgno);
    exit(2999);
  }

  /* print out the current settings...underflow should be enabled */
  printf("%.80s\n",cond_string);

  /* pop the previous setting off the stack */
  action = 4;
  CEE3SPM(&action,cond_string,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
    printf("CEE3SPM pop failed with message %\n",fc.tok_msgno);
    exit(2999);
  }
}
```

Example Using Several Condition Handling Services

Figure 79 on page 324 contains a C/370 example in which a user signal handler is registered and a condition is signalled to the LE/370 condition manager. The user signal handler gains control and calls CEEGQDT to access a data structure that was set up in the signalling routine. The CEEMRCR callable service is used to reset the resume cursor, and execution is resumed at the new point.

2. C/370 Example —

```
/*********************************************************************/
/* This example checks the LE/370 Hardware Condition Enablement and   */
/* if the UNDERFLOW condition is not enabled it is enabled.           */
/* The settings are then reset to the original values.                */
/*********************************************************************/
#include <leawi.h>
#include <stdio.h>
#include <string.h>

#define SUCCESS "\0\0\0\0"

int main(void) {

  _FEEDBACK fc;
  _INT4 action;
  _CHAR80 cond_string;
  char *cond;

  /* query the current settings */
  action = 2;
  CEE3SPM(&action,cond_string,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEE3SPM query failed with message %\n",fc.tok_msgno);
     exit(2999);
   }

  /* enable the underflow condition if it is not enabled */

  cond = strstr(cond_string,"NOU");
  if (cond != NULL) {
     memcpy(cond,"U  ",3);
     action = 5;              /* PUSH and SET              */
  }
  else
     action = 3;              /* PUSH                      */

  /* enable the underflow condition if necessary          */
  CEE3SPM(&action,cond_string,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEE3SPM failed with message %\n",fc.tok_msgno);
     exit(2999);
   }

  /* query the current settings */
  action = 2;
  CEE3SPM(&action,cond_string,&fc);
```

4. Some S/370 hardware interrupt codes and their matching LE/370 feedback codes can be found in Table 47 on page 321.

Table 47. S/370 Interrupt codes

S/370 inter code	Description	Maskable	Symbolic LE FBCode	Message number	Severity
0001	operation exception	no	CEE341	3201	3
0002	privileged operation exception	no	CEE342	3202	3
0003	execute exception	no	CEE343	3203	3
0004	protection exception	no	CEE344	3204	3
0005	addressing exception	no	CEE345	3205	3
0006	specification exception	no	CEE346	3206	3
0007	data exception	no	CEE347	3207	3
0008	fixed point overflow exception	yes	CEE348	3208	3
0009	fixed point divide exception	no	CEE349	3209	3
000A	decimal overflow exception	yes	CEE34A	3210	3
000B	decimal divide exception	no	CEE34B	3211	3
000C	exponent overflow exception	no	CEE34C	3212	3
000D	exponent underflow exception	yes	CEE34D	3213	3
000E	significance exception	yes	CEE34E	3214	3
nn0F	floating point divide exception	no	CEE34F	3215	3

Examples

1. COBOL/370 Example —

```
77   ACTION PIC S9(9) COMP.
77   CONDSTR PIC X(80).
77   FC PIC X(12).
:

     MOVE 2 TO ACTION.
     CALL "CEE3SPM" USING ACTION , CONDSTR , FC.
```

A list of conditions that can be enabled and disabled and their associated identifiers is given below:

Condition	Identifier
FIXED-OVERFLOW	F (NOF for disablement)
DECIMAL-OVERFLOW	D (NOD for disablement)
UNDERFLOW	U (NOU for disablement)
SIGNIFICANCE	S (NOS for disablement)

An identifier with the "NO" prefix is used to disable the condition it represents. An identifier without the "NO" prefix is used to enable the condition that it represents. For example, the token "F" is used to enable the FIXED-OVERFLOW condition. The identifier "NOF" is used to disable the FIXED-OVERFLOW condition. The rightmost option takes effect in the event of a conflict. Identifiers are separated by blanks or commas.

fc (output)
an optional 12-byte condition token that indicates the results of the service. Not all feedback codes are valid for each *action* requested.

The following symbolic conditions may result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE36V	4	3295	The condition string from CEE3SPM did not contain all of the settings, because the returned string was truncated.
CEE370	4	3296	Some of the data in the condition string from CEE3SPM could not be recognized.
CEE371	4	3297	The service completed successfully for recognized condition(s), unsuccessfully for unrecognized (unsupported) condition(s).
CEE372	4	3298	A call to CEE3SPM attempted to PUSH settings onto a full stack.
CEE373	4	3299	A call to CEE3SPM attempted to POP settings off an empty stack.
CEE374	4	3300	The action parameter in CEE3SPM was not one of the digits 1 to 5.

Usage Notes

1. When you use the CEE3SPM callable service, maintenance of the condition stack is required.

   For example, one user-written condition handler can disable a hardware condition while another enables it. Therefore, do not assume that the program mask is at a given setting.

2. The program mask is set differently based on different HLL requirements. You can find out what the current setting is by using query function CEE3SPM.

3. LE/370 initialization sets conditions based on the languages in the initial load module. Each language present adds to the conditions that are enabled.

- Push the current settings of the LE/370 conditions onto the LE/370-managed condition stack and alter the settings of the LE/370 conditions to those supplied by the caller.

The LE/370 conditions that can be enabled or disabled are listed below:

FIXED-OVERFLOW When enabled, raises the FIXED-OVERFLOW condition when an overflow occurs during signed binary arithmetic or signed left-shift operations.

DECIMAL-OVERFLOW When enabled, raises the DECIMAL-OVERFLOW condition when one or more nonzero digits are lost because the destination field in a decimal operation is too short to contain the results.

UNDERFLOW When enabled, raises the UNDERFLOW condition when the result characteristic of a floating-point operation is less than zero and the result fraction is not zero. For an extended-format floating-point result, the condition is raised only when the high-order characteristic underflows.

SIGNIFICANCE When enabled, raises the SIGNIFICANCE condition when the resulting fraction in floating-point addition or subtraction is zero.

Syntax

```
▸▸——CEE3SPM—(—action—,—cond_string——,—┬────┬—)——————————▸◂
                                      └─fc─┘
```

action (input)
is the action to be performed. *action* is specified as a fullword signed integer corresponding to one of the numbers in the following list:

Action	Description
1	SET - alter the settings of the LE/370 conditions to those specified in the *cond_string* parameter
2	QUERY - query the current settings of the LE/370 conditions and return the settings in the *cond_string* parameter
3	PUSH - push the current settings of the LE/370 conditions onto the LE/370-managed condition stack
4	POP - pop the pushed settings of the LE/370 conditions from the LE/370-managed condition stack and set the current settings of the LE/370 conditions to those popped.
5	PUSH and SET - push the current settings of the LE/370 conditions onto the LE/370-managed condition stack and alter the settings of the LE/370 conditions to those supplied by the caller in the *cond_string* parameter.

cond_string (input/output)
a fixed length string of 80 bytes containing a sequence of identifiers that represent the requested settings for the LE/370 conditions that can be enabled and disabled.

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#define SUCCESS "\0\0\0\0"

int main(void) {

  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;

  .
  .
  .
  /* register condition handler */
  CEEHDLR(&routine,&token,&fc);
  .
  .
  .
  /* signal condition */
  CEESGL(&condtok,&qdata,NULL);
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result, _FEEDBACK *newfc) {

   _CHAR80 name;

   /* get name of the routine that signal the condition */
   CEE3GRN(name,NULL);

   printf("the routine that called this condition handler is:\n");
   printf("%.80s\n",name);
   *result = 10;
   return;
}
```

CEE3SPM — Query and Modify LE/370 Hardware Condition Enablement

The CEE3SPM callable service is used to query and modify the enablement of LE/370 hardware conditions. You can use the CEE3SPM service to:

- Alter the settings of the LE/370 conditions to those specified by the caller
- Query the current settings of the LE/370 conditions and return the settings to the caller
- Push the current settings of the LE/370 conditions onto the LE/370-managed condition stack, where all program math settings are kept
- Pop the pushed settings of the LE/370 conditions from the LE/370-managed condition stack and set the current settings of the LE/370 conditions to those popped

Possible conditions returned in *fc* are:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE35S	1	3260	No condition was active when a call to a condition management routine was made.

Examples

1. COBOL/370 Example —

```
77   RNAME PIC X(80).
77   FC PIC X(12).
:

     CALL "CEE3GRN" USING RNAME , FC.
```

Examples

1. COBOL/370 Example —

```
        77   ALLOW PIC S9(9) COMP.
        77   FC PIC X(12).
  :

            MOVE 0 TO ALLOW.
            CALL "CEE3CNC" USING ALLOW , FC.
```

2. C/370 Example —

```
/* user condition handler */

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {

  _INT4 allow;

  /* allow nested conditions */
  allow = 1;
  CEE3CNC(&allow,NULL);
  .
  .
  .
}
```

CEE3GRN —Get Name of Routine that Incurred Condition

The CEE3GRN callable service obtains the name of the most current LE/370-enabled routine where a condition occurred. If there are nested conditions, the most recently signaled condition is used.

Syntax

```
▸▸──CEE3GRN──(──name──,──┬────┬──)──────────────────▸◂
                         └─fc─┘
```

name (output)

In the absence of a descriptor, this field is a fixed-length 80 character string that contains the name of the routine that was executing when the condition was raised. The *name* is left-justified within the field and right-padded with blanks. If there are nested conditions, the most recently activated condition is used to determine the *name*.

fc (output)

an optional 12-byte condition token that indicates the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

2. C/370 Example —

```
#include <leawi.h>

int main(void) {

  _INT4 code, timing;

  code = 3415;         /* HEX value DEAD */
  timing = 0;

  CEE3ABD(&code,&timing);
}
```

CEE3CNC —Allow Nested Conditions

The CEE3CNC callable service allows or prohibits nested conditions within a condition handler that you registered using the CEEHDLR callable service.

Syntax

```
▸▸──CEE3CNC─(──allow──,─┬────┬─)──────────────────────────▸◂
                        └─fc─┘
```

allow (input)
a fullword integer.

If you specify a non-zero value for *allow*, nested conditions are allowed. If you specify zero as the *allow* value, nested conditions will not be tolerated.

fc (output)
an optional 12-byte condition token that indicates the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

Possible conditions returned in *fc* are:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE35S	1	3260	No condition was active when a call to a condition management routine was made.

Syntax

```
▸▸——CEE3ABD—(—abcode—,—timing—)——————————————————▸◂
```

abcode

a fullword integer, no greater than 4095, specifying the ABEND code that is issued. When executing under CICS, this fullword integer will be converted to the equivalent EBCDIC and then issued.

timing

a fullword integer specifying that the ABEND should be issued immediately. The only accepted value for timing is 0, meaning issue the ABEND immediately upon return without any clean-up.

When an immediate ABEND is requested, no termination activities are performed:

- Event handlers are not driven.
- AD/Cycle CODE/370 is not invoked.
- User exits are not invoked.
- User written condition handlers registered by CEEHDLR are not invoked.

Usage Notes

1. There is no return from this service, nor is there any condition associated with it.
2. A system dump is requested when the ABEND is issued. The dump request is not honored under CMS.

Examples

1. COBOL/370 Example —

```
77   ABDCODE PIC S9(9) COMP.
77   TIMING PIC S9(9) COMP.
 :

     MOVE 3415 TO ABDCODE.
     MOVE 0 TO TIMING.

     CALL "CEE3ABD" USING ABDCODE , TIMING.
```

Examples

1. COBOL/370 Example —

```
77   CONDTOK PIC X(12).
77   QDATA PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE 0 TO QDATA.
     CALL "CEESGL" USING CONDTOK , QDATA , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#define SUCCESS "\0\0\0\0"

int main(void) {

  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;
  .
  .
  .
  /* build the condition token */
  condtok.tok_sev = 1;
  condtok.tok_msgno = 99;
  condtok.tok_case = 1;
  condtok.tok_sever = 1;
  condtok.tok_ctrl = 0;
  memcpy(condtok.tok_facid,"ZZZ",3);
  condtok.tok_isi = 0;
  qdata = 0;

  /* signal the condition */
  CEESGL(&condtok,&qdata,NULL);
  .
  .
  .
}
```

CEE3ABD —Terminate Enclave with an ABEND

The CEE3ABD callable service terminates the enclave immediately with an ABEND.

cond_rep (input)
a 12-byte condition token representing the condition to be signaled.

You can either construct your own condition token or use one that LE/370 has already defined. To construct your own condition token, see "CEENCOD — Construct a Condition Token" on page 306.

q_data_token (input/output)
an optional 32-bit data object that is placed in the ISI for use in accessing the math condition qualifying data associated with the given instance of the condition. The qualifying data is a list of information addresses that a math condition handler can use to specifically identify and, if necessary, react to, a given condition. The information in the math q_data, therefore, provides a mechanism by which math user-written condition handlers can provide a complete fix-up.

fc (output)
an optional 12-byte condition token that indicates the result of the CEESGL service.

Conditions returned in *fc*:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE069	0	0201	An unhandled condition was returned in a feedback code.
CEE0CE	1	0398	Resume with new input.
CEE0CF	1	0399	Resume with new output.
CEE0EE	3	0462	Instance specific information for the condition token with message number *message-number* and facility ID *facility-id* could not be found.

Usage Notes

1. Unique conditions signaled by CEESGL are considered *enabled* under LE/370.
2. Each enabled signaled condition (of severity 2 or above) increments the error count by one. If the error count exceeds the error count limit (as specified by the ERRCOUNT run-time option – see "ERRCOUNT" on page 229), the condition manager terminates the enclave without signaling Termination_Imminent and terminates with ABEND code 4091, reason code 11. Promoted conditions do not increment the error count.
3. Severity 0 and 1 conditions are considered safe conditions. They can be ignored if they are not handled and no feedback token was passed when the condition was raised.
4. Conditions signaled with this service are not necessarily handled by LE/370. If you call CEESGL with a *cond_rep*, LE/370 passes control to the appropriate HLL condition handler. The HLL then determines whether it should handle the condition. If so, the HLL handles the condition. If not, control returns to LE/370.
5. Table 47 on page 321 contains a list of the S/370 program interrupt codes and the corresponding LE/370 condition token name and message number.

```
/*********************************************************************/
/* In C/370 it is not necessary to use this service.  The fields can */
/* be manipulated directly.  See the example for CEESGL to see how to */
/* manipulate condition token fields directly.                       */
/*********************************************************************/

#include <leawi.h>
#include <stdio.h>
#include <string.h>

#define SUCCESS "\0\0\0\0"

int main(void) {

  _FEEDBACK fc,condtok;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;

  c_1 = 1;
  c_2 = 99;
  cond_case = 1;
  sev = 1;
  control = 0;
  memcpy(facid,"ZZZ",3);
  isi = 0;

  CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,facid,&isi,&condtok,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEENCOD failed with message number %d\n",fc.tok_msgno);
     exit(2999);
  }
  .
  .
  .
}
```

CEESGL — Signal a Condition

The callable service CEESGL raises, or signals, a condition to the condition manager. It also can be used to provide qualifying data and to create an ISI (Instance Specific Information) for this particular instance of the condition. The ISI contains information that is used by the LE/370 condition manager to identify and react to conditions.

LE/370 defines a set of signals, but CEESGL can also be used to signal user-defined conditions.

Syntax

```
▸▸──CEESGL──(──cond_rep──,─┬──────────────────┬─┬────┬─)──────────▸◂
                           └─q_data_token,─┘   └─fc─┘
```

```
77   SEV PIC S9(4) COMP.
77   MSGNO PIC S9(4) COMP.
77   CASE PIC S9(4) COMP.
77   SEV2 PIC S9(4) COMP.
77   CNTRL PIC S9(4) COMP.
77   FACID PIC X(3).
77   ISINFO PIC S9(9) COMP.
77   NEWTOK PIC X(12).
77   FC PIC X(12).
 :

     MOVE 0 TO SEV.
     MOVE 1 TO MSGNO.
     MOVE 1 TO CASE.
     MOVE 0 TO SEV2.
     MOVE 1 TO CNTRL.
     MOVE "CEE" TO FACID.
     MOVE 0 TO ISINFO.
     CALL "CEENCOD" USING SEV , MSGNO , CASE , SEV2 , CNTRL ,
     FACID , ISINFO , NEWTOK , FC.
```

2. C/370 Example —

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0CH	3	0401	An unsupported case code *case-code* was passed to routine *routine-name.*
CEE0CI	3	0402	An unsupported control code *control-code* was passed to routine *routine-name.*
CEE0CJ	3	0403	An unsupported severity code *severity-code* was passed to routine *routine-name.*
CEE0CK	1	0404	A facility ID *facility-id* with non-alphanumeric characters was passed to routine *routine-name.*
CEE0E4	3	0452	An unsupported facility ID *facility-id* was passed to routine *routine-name.*

Usage Notes

1. The Condition_ID is a fullword identifier that, with the *Facility_id* parameter of CEENCOD, describes the condition that the token communicates. The Condition_ID contains two case fields; these differ depending on whether they are case 1 or case 2 types. These cases represent the types of conditions that the condition token will communicate.
2. See Figure 34 on page 79 contains a mapping of a condition token layout.
3. C/370 Considerations — The structure of the condition token (type_FEEDBACK) is described in the leawi.h header file shipped with LE/370. C/370 users can assign values directly to the fields of the token in the header file without using the CEENCOD service.

   The layout of the type_FEEDBACK condition token in the header file is shown in Figure 78.

```
typedef struct {
   short      tok_sev     ;            /* severity                  */
   short      tok_msgno   ;            /* message number            */
   int        tok_case :2,             /* flags - case/sev/control  */
              tok_sever:3,
              tok_ctrl :3 ;
   char       tok_facid[3];  /* facility ID             */
   int        tok_isi     ;            /* index into ISI block      */
        }                 _FEEDBACK;
```

Figure 78. type_FEEDBACK Data Type as Defined in the leawi.h Header File

Examples

1. COBOL/370 Example —

1 indicates that the facility_id has been assigned by IBM. 0 indicates that the facility_id has been assigned by the user.

Facility_ID (input)
a 3-character field containing three alphanumeric characters that identify the product generating this condition or feedback information, for example, 'CEE'. Special characters, including blank spaces, cannot be used.

The *Facility_ID* is associated with the repository (for example, a file) of the run-time messages. If a unique ID is required (for IBM and non-IBM products), an ID can be obtained by contacting an IBM project office.

A *Facility_ID* assigned by IBM to an IBM product must begin with one of the letters A through I, inclusive. A *Facility_ID* assigned by IBM to a product other than an IBM product must not begin with a letter A through I. See the section entitled **Control**, above, on how to indicate if the Facility_ID has been assigned by IBM. There are no other constraints (besides the alphanumeric requirement) on a *Facility_ID* not assigned by IBM.

I_S_Info (input)
a fullword binary integer identifying the Instance Specific Information (ISI)

Whenever a condition is detected by the LE/370 condition manager, insert data which describes the particular instance of its occurrence is generated. This insert data may be used, for example, to write to a file a message associated with that particular instance or occurrence of the condition. This insert data is contained in the ISI.

Cond_Token (output)
the 12-byte representation of the constructed condition token.

fc (output)
an optional 12-byte condition token that indicates the result of the CEENCOD service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

The following symbolic conditions can result from this service:

contains:

Severity
a 2-byte binary integer specifying the severity of the condition. You can specify the Severity in the *C_1* parameter of CEENCOD.

Msg_no
a 2-byte binary number that identifies the message associated with the condition. You can specify the Msg_no in the *C_2* parameter of CEENCOD.

Case 2 - Class/Cause Code Condition
Case 2 is used by some operating systems and compiler run-time libraries to modify a particular message or condition. It contains:

Class_Code
a 2-byte, binary number that identifies the message id (same type as **severity** above) associated with the **class** of the condition. You can specify Class_Code in the *C_1* parameter of CEENCOD.

Cause_Code
a 2-byte, binary number that identifies the message subid (same type as Msg_No, above) associated with the **cause** of the condition. You can specify Cause_Code in the *C_2* parameter of CEENCOD.

Note: The Class_Code/Cause_Code fields are assigned by products other than CEE.

Case (input)
a 2-byte binary integer that defines the format of the Condition_ID portion of the token. The value 1 identifies a case 1 condition, a value of 2 identifies a case 2 condition. The values 0 and 3 are reserved.

Severity (input)
a 2-byte binary integer that indicates the condition's severity. For case 1 conditions, the value of this field is the same as the severity value specified in the Condition_ID.

For case 1 and 2 conditions, this field may always be used to test the condition's severity. *Severity* can be specified with the following values:

0	Information only (or, if the entire token is zero, no information)
1	Warning - service completed, probably correctly
2	Error detected - correction attempted; service completed, perhaps incorrectly
3	Severe error - service not completed
4	Critical error - service not completed; condition signaled. A critical error is a condition that jeopardizes the environment. If a critical error occurs during an LE/370 callable service, it is always signaled to the condition manager instead of being returned synchronously to the caller.

Control (input)
a 2-byte binary integer containing flags describing or controlling various aspects of condition handling. Valid values for the control field are 1 and 0.

```
   /* move the resume cursor to the caller of the routine */
   /* that registered the condition handler               */
   type = 1;
   CEEMRCR(&type,&cursorfc);
   if (memcmp(&cursorfc,SUCCESS,4) != 0) {
      printf("CEEMRCR failed with message number %d\n",cursorfc.tok_msgno);
      exit (2999);
   }

   printf("condition handled\n");
   *result = 10;
   return;
}
```

CEENCOD — Construct a Condition Token

A condition in the common run-time environment is communicated by a 12-byte condition token. The CEENCOD callable service is used to dynamically construct one of these condition tokens.

The condition token communicates with message services, condition management, LE/370 callable services, and user applications. All common run-time environment callable services also use the condition token data type to return information to the user of the service as a feedback code. Figure 34 on page 79 contains a mapping of the layout of the condition token.

C/370 users You can alter the fields of the condition token without having to use the CEENCOD service. See the example for "CEESGL — Signal a Condition" on page 311 for details.

Syntax

```
►►──CEENCOD──(──C_1──,──C_2──,──Case──,──Severity──,──Control──────►

►──Facility_ID──,──I_S_Info──,──Cond_Token──,─┬────┬─)─────────►◄
                                              └─fc─┘
```

C_1 (input)

C_1 and *C_2* together comprise the Condition_ID portion of the condition token. *C_1* is a 2-byte binary integer representing the value of the first 2 bytes of the 4-byte Condition_ID.

For case 1, *C_1* represents the Severity; for case 2, it is the Class_Code. For more information about case 1 and case 2, see the Usage Notes for this service.

C_2 (input)

a 2-byte binary integer representing the value of the second 2 bytes of the Condition_Id.

For case 1, this is the Msg_No; for case 2, it is the Cause_Code. For more information about case 1 and case 2, see CEENCOD Usage Notes.

Case 1 - Service Condition

Case 1 is used by all LE/370 callable services and most applications. It

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#define SUCCESS "\0\0\0\0"

int main(void) {

  .
  .
  .
  b();
  /* the CEEMRCR call in the handler will place the resume cursor at */
  /* this point.                                                     */
  .
  .
  .
}


void b(void) {

  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;

  /* register the condition handler */
  CEEHDLR(&routine,&token,&fc);
  .
  .
  .
  /* signal the condition */
  CEESGL(&condtok,&qdata,NULL);
  .
  .
  .
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {

   _FEEDBACK cursorfc, orig_fc;
   _INT4 type;
   .
   .
   .
```

In Figure 77, the user condition handlers are established for routines C and D. When a condition is raised in routine D, only a *type_of_move* = 1 is permitted. A 0 *type_of_move* results in CEE error message 254 and the application will ABEND.

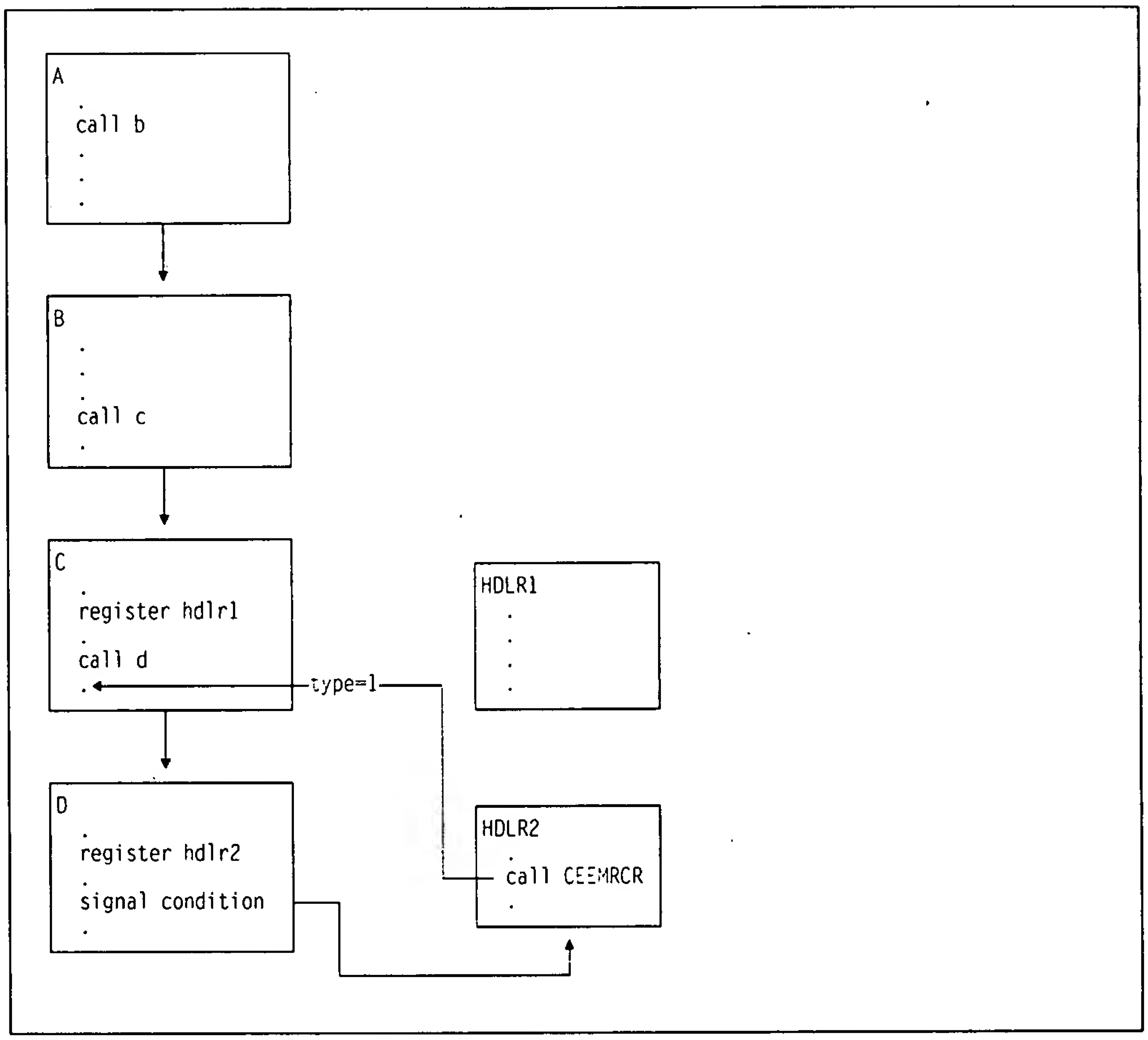


Figure 77. Moving Resume Cursor using CEEMRCR

Examples

1. COBOL/370 Example —

```
77   MOVETYP PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE 0 TO MOVETYP.
     CALL "CEEMRCR" USING MOVETYP , FC.
```

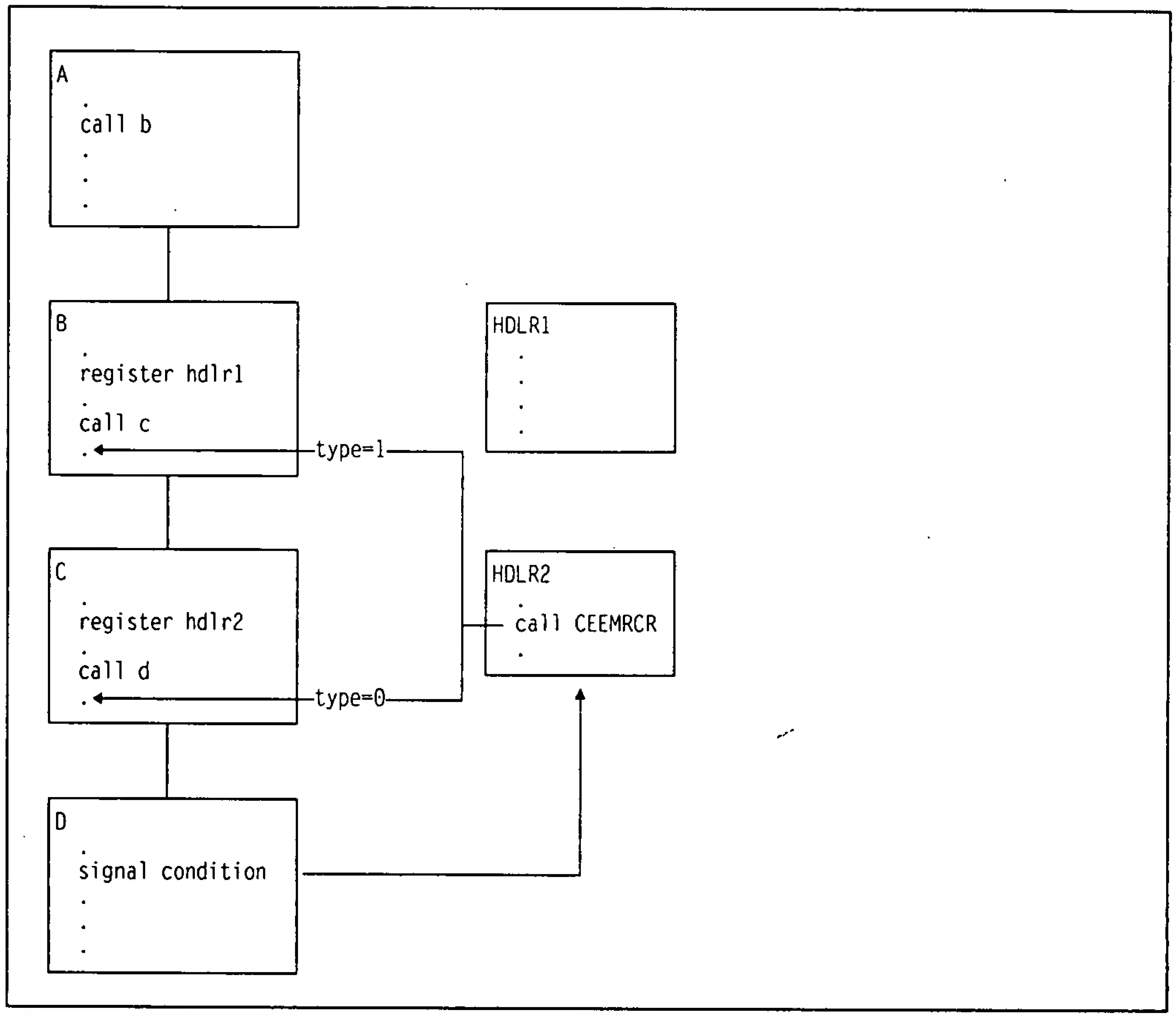


Figure 76. Moving Resume Cursor using CEEMRCR

stack frame immediately preceding the one to which the handle cursor points, moves the resume cursor to the instruction immediately following a call in routine A to routine B.

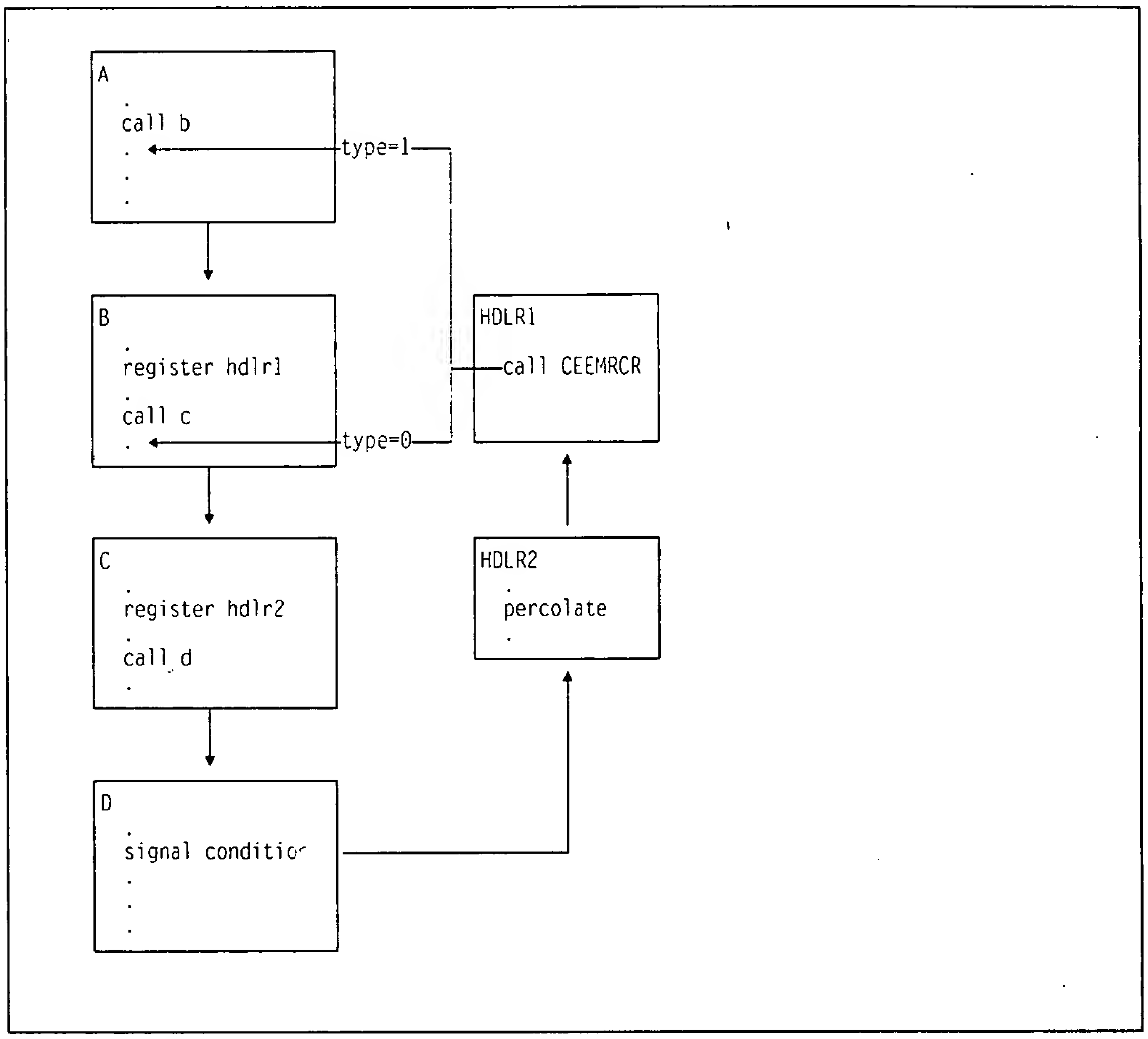


Figure 75. Moving Resume Cursor using CEEMRCR

The same scenario is illustrated in Figure 76 on page 303, except that HDLR2 issues a resume for the signalled condition rather than percolating it. HDLR1 never gains control. Since the handle cursor now points at the stack frame for routine C, a 0 *type_of_move* causes control to resume at the call return point in Routine C, the instruction immediately following the call to routine D. A 1 *type_of_move* moves the resume cursor to the instruction immediately following a call in routine B to routine C.

of movement is always toward older stack frames and never toward newer stack frames. The action occurs only after the condition handler has returned to the LE/370 condition manager.

Syntax

```
▸▸——CEEMRCR—(—type_of_move—,—┬────┬—)——————————————————▸◂
                             └─fc─┘
```

type_of_move (input)

a fullword signed integer indicating the target of the resume cursor movement. The possible values for *type_of_move* are:

0 Move the resume cursor to the call return point of the stack frame associated with the handle cursor.

1 Move the resume cursor to the call return point of the stack frame that precedes the stack frame associated with the handle cursor. The handle cursor is moved to the first condition handler of the stack frame that the new resume cursor position now points to.

fc (output)

an optional 12-byte condition token that indicates the result of the service. Conditions returned in *fc* are:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE07U	1	0254	The first argument passed to CEEMRCR was not 0 or 1.
CEE080	1	0256	The routine specified was already registered for this stack frame. It was registered again.

Usage Notes

1. Multiple calls to CEEMRCR yield the NET results of the calls; that is, if two calls move the resume cursor to different places for the same stack frame, the most restrictive call (that closest to stack frame 0) is used for that stack frame.

2. The following three figures illustrate moving the resume cursor using the CEEMRCR service.

   In Figure 75 on page 302, routine A calls routine B, which in turn calls C, which calls D. User condition handlers are registered in routines B and C. When a condition is signaled in routine D, the LE/370 condition manager passes control to the user condition handler established for routine C. The handle cursor now points at the stack frame for routine C. Routine C percolates the condition. The handle cursor now points at the stack frame for routine B. The next user condition handler to gain control, that established for Routine B, recognizes the condition and issues a resume by calling CEEMRCR. A 0 *type_of_move*, meaning move the resume cursor to the stack frame associated with the handle cursor, causes control to resume at the call return point in Routine B, the instruction immediately following the call to routine C. A 1 *type_of_move*, meaning move the resume cursor to the call return point of the

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#define SUCCESS "\0\0\0\0"

int main(void) {

  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;

  /* register condition handler */
  CEEHDLR(&routine,&token,&fc);
  .
  .
  .
  /* signal the condition */
  CEESGL(&condtok,&qdata,NULL);
  .
  .
  .
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {

   _FEEDBACK orig_fc;
   .
   .
   .
   /* get the original condition token */
   CEEITOK(&orig_fc,NULL);
   .
   .
   .
}
```

CEEMRCR — Move Resume Cursor Relative to Handle Cursor

The CEEMRCR callable service moves the resume cursor to a position relative to the current position of the handle cursor. The actions supported are:

- Move the resume cursor to the call return point of the routine that registered the executing condition handler.
- Move the resume cursor to the caller of the routine that registered the executing condition handler.

Initially, the resume cursor is placed after the machine instruction that caused the condition. Whenever the resume cursor is moved, as each stack frame is passed, any associated exit routine is invoked. Note that "exit routine" refers to user condition handlers as well as language-specific condition handlers. This moving also unregisters any associated user condition handlers. The direction

Syntax

```
▸▸——CEEITOK—(—I_CTOK—,—┬────┬—)——————————————————▸◂
                       └─fc─┘
```

I_CTOK(output)
: a 12-byte condition token (see "CEENCOD — Construct a Condition Token" on page 306 for information) that identifies the initial condition in the current active data block that is being processed.

fc (output)
: an optional 12-byte condition token that indicates the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

Possible conditions returned in *fc* are:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE35S	1	3260	No condition was active when a call to a condition management routine was made.

Examples

1. COBOL/370 Example —

```
77   ITOKEN PIC X(12).
77   FC PIC X(12).
:

     CALL "CEEITOK" USING ITOKEN , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#define SUCCESS "\0\0\0\0"

int main(void) {

  _FEEDBACK fc;
  _ENTRY routine;
  _INT4 token;

  /* set the routine structure to point to the handler */
  /* and use CEEHDLR to register the user handler      */

  token = 99;
  routine.address = (_POINTER)&handler;
  routine.nesting = NULL;

  CEEHDLR(&routine,&token,&fc);

  /* verify that CEEHDLR was successful */
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEHDLR failed with message number %d\n",fc.tok_msgno);
     exit (2999);
  }
  .
  .
  .
  /* Unregister the condition handler */
  CEEHDLU(&routine,&fc);

  /* verify that CEEHDLU was successful */
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEHDLU failed with message number %d\n",fc.tok_msgno);
     exit (2999);
  }
  .
  .
  .
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {
  .
  .
  .
}
```

CEEITOK — Return Initial Condition Token

The CEEITOK service returns the initial condition for the current "Condition Information Block", the platform specific data block used by the LE/370 condition manager as a repository for data about conditions raised in the LE/370 run-time environment.

CEEHDLR by the same stack frame that is invoking CEEHDLU. See "CEEHDLR — Register User Condition Handler" on page 294 for further information about specifying the *routine* parameter.

fc (output)
an optional 12-byte condition token that indicates the result of the service.

Conditions returned in *fc*:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE07S	1	0252	CEEHDLU was unable to find the requested user condition handler routine.

Usage Note

1. You are not required to remove user condition handlers that you have registered through the CEEHDLR callable service. Any user condition handlers created through CEEHDLR and not unregistered by CEEHDLU are unregistered automatically by LE/370 when the associated stack frame is removed from the stack.

Examples

1. COBOL/370 Example —

```
77   ROUTINE  PROCEDURE-POINTER.
77   FC PIC X(12).
 :

     SET ROUTINE TO ENTRY "HANDLER".
     CALL "CEEHDLU" USING ROUTINE , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#define SUCCESS "\0\0\0\0"

int main(void) {

  _FEEDBACK fc;
  _ENTRY routine;
  _INT4 token;


  /* set the routine structure to point to the handler */
  /* and use CEEHDLR to register the user handler      */

  token = 99;
  routine.address = (_POINTER)&handler;
  routine.nesting = NULL;

  CEEHDLR(&routine,&token,&fc);

  /* verify that CEEHDLR was successful */
  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEEHDLR failed with message number %d\n",fc.tok_msgno);
     exit (2999);
  }
  .
  .
  .
}
/*******************************************************************/
/* handler is a user condition handler                             */
/*******************************************************************/
void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {
  .
  .
  .
}
```

CEEHDLU — Unregister User Condition Handler

The CEEHDLU callable service unregisters a user condition handler for the current stack frame.

Syntax

```
▸▸——CEEHDLU—(—routine—,——————— ——————————————————▸◂
                         └—fc—┘
```

routine (input)

an entry variable or constant for the routine that is to be unregistered as a user condition handler. This routine must have been previously registered using

Conditions returned in *fc* are:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE080	1	0256	The routine specified was already registered for this stack frame. It was registered again.
CEE081	3	0257	The routine specified contained an invalid entry variable.

Usage Notes

1. LE/370 places the user condition handlers associated with each stack frame in a queue. The queue can be empty at any given time. The LE/370 condition manager invokes the registered condition handlers in LIFO (last in, first out) order to handle the condition.

2. The counterpart of CEEHDLR, which registers a user condition handler, is CEEHDLU, which unregisters the handler. However, you are not required to remove user condition handlers that you have registered through the CEEHDLR callable service. Any user condition handlers created through CEEHDLR and not unregistered by CEEHDLU are unregistered automatically by LE/370 when the associated stack frame is removed from the stack.

Examples

1. COBOL/370 Example —

```
77   ROUTINE  PROCEDURE-POINTER.
77   TOKEN PIC S9(9) COMP.
77   FC PIC X(12).
:

     SET ROUTINE TO ENTRY "HANDLER".
     CALL "CEEHDLR" USING ROUTINE , TOKEN , FC.
```

```
void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {

   _FEEDBACK qdatafc;
   _INT4 qdata;
   .
   .
   .
   /* get the q_data_token from the ISI */
   CEEGQDT(fc,&qdata,&qdatafc);

   if (memcmp(&qdatafc,SUCCESS,4) != 0) {
      printf("CEEGQDT failed with message number %d\n",qdatafc.tok_msgno);
      exit(2999);
   }
   /* use the condition info structure (qdata) */
   if (qdata->error_value == 86) {
      printf("%.12s\n",info->error_msg);
      exit(info->retcode);
   }
   .
   .
   .
}
```

CEEHDLR — Register User Condition Handler

The CEEHDLR callable service is used to register a user condition handler for the current stack frame (note that stack frames are called by various names in different HLLs — see "Obtaining the Stack Frame" on page 83 for more information). The user condition handler is invoked when:

- It is registered for the current stack frame by CEEHDLR, and
- The LE/370 condition manager requests that the condition handler associated with the current stack frame handles the condition.

Syntax

```
▸▸──CEEHDLR──(──routine──,──token──,──┬────┬──)──────────────▸◂
                                      └─fc─┘
```

routine (input)
an entry variable or entry constant for the routine that is to be 'called' to process the condition. The entry variable or constant must be passed by reference. The routine must be an external routine; that is, it must not be a nested routine.

token (input)
a fullword integer of information that you want to be passed to your user handler each time that it is called. This can be a pointer or any other fullword integer you wish to pass.

fc (output)
an optional 12-byte condition token that indicates the result of the service.

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#define SUCCESS "\0\0\0\0"

typedef struct {          /* condition info structure */
   int   error_value;
   char err_msg[80];
   int   retcode;
} info_struct;

info_struct *info;


int main(void) {

   _FEEDBACK fc,condtok;
   _ENTRY routine;
   _INT4 token,qdata;
   .
   .
   .
   /* register the condition handler */
   CEEHDLR(&routine,&token,&fc);
   .
   .
   .
   /* set up the condition info structure */
   info = malloc(sizeof(info_struct));
   if (info == NULL) {
      printf("error allocating info_struct\n");
      exit(2399);
   }

   info->error_value = 86;
   memcpy(info->error_msg,"Test message",12);
   info->retcode = 99;

   /* set qdata to be the condition info structure */
   qdata = (int)info;

   /* signal the condition */
   CEESGL(&condtok,&qdata,NULL);
   .
   .
   .
}
```

to the Condition Manager.
fc may return the following conditions:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0EE	3	0462	Instance specific information for the condition token with message number *message-number* and facility ID *facility-id* could not be found.
CEE0EG	3	0464	Instance specific information for the condition token with message number *message-number* and facility ID *facility-id* did not exist.

Examples

1. COBOL/370 Example —

```
77   CONDTOK PIC X(12).
77   QDATA PIC S9(9) COMP.
77   FC PIC X(12).
:

     CALL "CEEGQDT" USING CONDTOK , QDATA , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

#define SUCCESS "\0\0\0\0"

int main(void) {

  _INT4 cee_ver_id, plat_id;
  _FEEDBACK fc;

  /* get the LE version and the platform id */
  CEEGPID(&cee_ver_id,&plat_id,NULL);

  printf("the LE/370 version is %d",cee_ver_id);
  printf(" the current platform is ");
  switch (plat_id) {
     case 2: printf("OS/2\n");
             break;
     case 3: printf("MVS/VM/370\n");
             break;
     case 4: printf("AS/400\n");
             break;
    default: printf("unrecognized platform id\n");
  }
}
```

CEEGQDT — Retrieve Q_Data_Token

The CEEGQDT callable service provides a mechanism by which application code (in particular, user condition handlers) can retrieve the *q_data_token* from the Instance Specific Information (ISI). The ISI contains information that is used by the LE/370 condition manager to identify and react to conditions. For more information, see the *q_data_token* parameter of the callable service "CEESGL — Signal a Condition" on page 311.

Syntax

```
▸▸──CEEGQDT──(──cond_rep──,──q_data_token──,──┬────┬──)──────────▸◂
                                              └─fc─┘
```

cond_rep (input)
a condition token that defines the condition how the q_data_token is retrieved.

q_data_token (output)
a 32-bit data object placed in the ISI service.

fc (output)
an optional 12-byte condition token that indicates the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument, and the requested operation was not successfully completed, the condition is signaled

CEEGPID —Retrieve the LE/370 Version and Platform ID

The CEEGPID callable service retrieves the LE/370 version ID and the platform ID of the platform and version of LE/370 that is in use for processing the currently active condition.

Syntax

```
▸▸───CEEGPID─(─CEE_Version_ID─,─Plat_ID─,─┬────┬─)──────────▸◂
                                          └─fc─┘
```

CEE_Version_ID(output)

a fullword integer representing the version of LE/370 that created this data block. The current value for this parameter is:

110 Version 1 Release 1 Modification level 0

Plat_ID(output)

a fullword integer representing the platform used for processing the current condition.

The current values of this parameter are:

2 OS/2*

3 MVS/VM/370

4 AS/400*

fc (output)

an optional 12-byte condition token that indicates the result of the service.

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the Condition Manager.

Possible conditions returned in *fc* are:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Examples

1. COBOL/370 Example —

```
77   VERSION PIC S9(9) COMP.
77   PLATID PIC S9(9) COMP.
77   FC PIC X(12).
:

     CALL "CEEGPID" USING VERSION , PLATID , FC.
```

Examples

1. COBOL/370 Example —

```
77   SEV PIC S9(4) COMP.
77   MSGNO PIC S9(4) COMP.
77   CASE PIC S9(4) COMP.
77   SEV2 PIC S9(4) COMP.
77   CNTRL PIC S9(4) COMP.
77   FACID PIC X(3).
77   ISINFO PIC S9(9) COMP.
77   CONTOK PIC X(12).
77   FC PIC X(12).
:

     CALL "CEEDCOD" USING CONTOK , SEV , MSGNO , CASE , SEV2 ,
     CNTRL , FACID , ISINFO , FC.
```

2. C/370 Example —

```
/***********************************************************************/
/* In C/370 it is not necessary to use this service.  The fields can  */
/* be manipulated directly.  See the example for CEESGL to see how to */
/* manipulate condition token fields directly.                        */
/***********************************************************************/

#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

  _FEEDBACK fc,newfc;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi, heapid, size;
  _POINTER address;

  heapid = 0;
  size = 4000;

  CEEGTST(&heapid,&size,&address,&fc);

  /* decompose the feedback token to check for errors */
  CEEDCOD(&fc,&c_1,&c_2,&cond_case,&sev,&control,facid,&isi,&newfc);

  if (c_1 != 0 || c_2 != 0) {
     printf("CEEGTST failed with msgno %d\n",c_2);
     exit(2999);
  }
  .
  .
  .
}
```

Control (output)
a 2-byte binary integer containing flags describing aspects of the state of the condition. Valid values for the control field are 1 and 0. 1 indicates that the facility_id has been assigned by IBM. 0 indicates that the facility_id has been assigned by the user. Thus, the returned values for the *control* field are similar to the *case* field.

Facility_ID (output)
a 3-character field containing three alphanumeric characters identifying the product generating the condition or feedback information. See the *Facility_ID* parameter of "CEENCOD — Construct a Condition Token" on page 306 for more information.

I_S_Info
a fullword binary integer identifying the Instance Specific Information (ISI) associated with the given instance of the condition represented by the condition token where it is contained. If an ISI is not associated with a given condition token, the I_S_Info field contains a value of binary zero.

fc (output)
an optional 12-byte condition token. If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. If not specified as an argument and the requested operation was not successfully completed, the condition is signaled to the condition manager.

The following symbolic conditions may result from this service:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0CH	3	0401	An unsupported case code *case-code* was passed to routine *routine-name*.
CEE0CI	3	0402	An unsupported control code *control-code* was passed to routine *routine-name*.
CEE0CJ	3	0403	An unsupported severity code *severity-code* was passed to routine *routine-name*.
CEE0CK	1	0404	A facility ID *facility-id* with non-alphanumeric characters was passed to routine *routine-name*.
CEE0E4	3	0452	An unsupported facility ID *facility-id* was passed to routine *routine-name*.

Usage Note

1. C/370 Considerations — The structure of the condition token (type_FEEDBACK) is described in the LEAWI.H header file shipped with LE/370. C/370 users can assign values directly to the fields of the token in the header file without using the CEENCOD service.

   The layout of the type_FEEDBACK condition token in the header file is shown in Figure 78 on page 309.

Chapter 35. Condition Handling

CEEDCOD — Decompose a Condition Token

The CEEDCOD callable service allows you to decompose or alter an existing condition token.

C/370 users You can decompose or alter the fields of the condition token without having to use the CEEDCOD service. See the example for "CEESGL — Signal a Condition" on page 311 for details.

Syntax

```
▶▶──CEEDCOD──(──Cond_Token──,──C_1──,──C_2──,──Case──,──────────────▶

▶──Severity──,──Control──,──Facility_ID──,──I_S_Info──,─┬────┬─)─▶◀
                                                        └─fc─┘
```

Cond_Token (input)
a 12-byte condition token representing the current condition or feedback information.

C_1 (output)
a 2-byte binary integer representing the value of the first 2 bytes of the Condition_ID. See "CEENCOD — Construct a Condition Token" on page 306 for an explanation of the Condition_ID.

C_2 (output)
a 2-byte binary integer representing the value of the second 2 bytes of the Condition_ID. See "CEENCOD — Construct a Condition Token" on page 306 for an explanation of the Condition_ID.

Case (output)
a 2-byte binary integer field that defines the format of the Condition_ID portion of the token. A value of 1 identifies a case 1 condition. A value of 2 identifies a case 2 condition. The values 0 and 3 are reserved. See "CEENCOD — Construct a Condition Token" on page 306 for an explanation of the Condition_ID.

Severity (output)
a 2-byte binary integer representing the severity of the condition. *Severity* can be specified with the following values:

0 Information only (or, if the entire token is zero, no information)

1 Warning - service completed, probably correctly

2 Error detected - correction attempted; service completed, perhaps incorrectly

3 Severe error - service not completed

4 Critical error - service not completed; condition signaled. A critical error is a condition that jeopardizes the environment. If a critical error occurs during an LE/370 callable service, it is always signaled to the condition manager, instead of being returned synchronously to the caller.

© Copyright IBM Corp. 1991

```
/* obtain storage from the new heap using CEEGTST */
CEEGTST(&heapid,&size,&address,&fc);

/* check the first 4 bytes of the feedback token (0 if successful) */
if (memcmp(&fc,SUCCESS,4) != 0) {
   printf("CEEGTST failed with message number %d\n",fc.tok_msgno);
   exit(99);
}

size = 200;  /* new size of heap element */

/* change the size of the heap element */
CEECZST(&address,&size,&fc);

/* check the first 4 bytes of the feedback token (0 if successful) */
if (memcmp(&fc,SUCCESS,4) != 0) {
   printf("CEECZST failed with message number %d\n",fc.tok_msgno);
   exit(99);
}

/* free the storage that was previously obtained using CEEGTST */
CEEFRST(&address,&fc);

/* check the first 4 bytes of the feedback token (0 if successful) */
if (memcmp(&fc,SUCCESS,4) != 0) {
   printf("CEEFRST failed with message number %d\n",fc.tok_msgno);
   exit(99);
}



/* discard the heap that was previously created using CEECRHP */
CEEDSHP(&heapid,&fc);

/* check the first 4 bytes of the feedback token (0 if successful) */
if (memcmp(&fc,SUCCESS,4) != 0) {
   printf("CEEDSHP failed with message number %d\n",fc.tok_msgno);
   exit(99);
}
}
```

Figure 74 (Part 2 of 2). C/370 example illustrating calls to several dynamic storage services

```
/***********************************************************************/
/* This example shows how several of the LE/370 Storage Management    */
/* Callable Services are used.  The services shown are:                */
/* CEE3RPH -- set the report heading                                   */
/* CEECRHP -- to create a user heap                                    */
/* CEEGTST -- to obtain a piece of storage from a heap                 */
/* CEECZST -- to change the size of a previously obtained piece of    */
/*            storage                                                  */
/* CEEFRST -- free a piece of storage obtained using CEEGTST           */
/* CEEDSHP -- discard a heap that was created by CEECRHP               */
/*                                                                     */
/* The example produces a storage report.                              */
/***********************************************************************/
#pragma runopts(REPORT)
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main (void) {

  _INT4 heapid, size, increment, options;
  _POINTER address;
  _FEEDBACK fc;
  _CHAR80 heading;
  #define SUCCESS "\0\0\0\0"

  /* use CEE3RPH to set the report heading */
  memset(heading,' ',80);
  memcpy(heading,"User Defined Report Heading",27);
  CEE3RPH(heading,NULL);


  heapid = 0;         /* heap identifier will be set by CEECRHP */
  size = 4096;        /* initial size of heap  (in bytes)       */
  increment = 4096;   /* increment to extend the heap by        */
  options = 71;       /* set up heap as (,,ANYWHERE,FREE)       */

  /* create heap using CEECRHP */
  CEECRHP(&heapid,&size,&increment,&options,&fc);

  if (memcmp(&fc,SUCCESS,4) != 0) {
     printf("CEECRHP failed with message number %d\n",fc.tok_msgno);
     exit(99);
  }

  size = 300;   /* number of bytes of heap storage */
```

Figure 74 (Part 1 of 2). C/370 example illustrating calls to several dynamic storage services

```
           MOVE 0 TO HEAPID.
           MOVE 0 TO HPSIZE.
           MOVE 0 TO INCR.
           MOVE 0 TO OPTS.
           MOVE 4000 TO NBYTES.

           CALL "CEECRHP" USING HEAPID , HPSIZE , INCR , OPTS , CRHPFC.
           IF CRHPFC = LOW-VALUE
              THEN DISPLAY "-+>PASS CE1DY010_1"
              ELSE DISPLAY "-->FAIL CE1DY010_1"
           END-IF.


           CALL "CEEGTST" USING HEAPID , NBYTES , ADDRSS , GTSTFC.
      *                 .
      *                 .
      *                 .
           CALL "CEEGTST" USING HEAPID , NBYTES , ADDRSS , GTSTFC.

           IF GTSTFC = LOW-VALUE
              THEN DISPLAY "-+>PASS CE1DY010_2"
              ELSE DISPLAY "-->FAIL CE1DY010_2"
           END-IF.


           CALL "CEEDSHP" USING HEAPID , DSHPFC.
           IF DSHPFC = LOW-VALUE
              THEN DISPLAY "-+>PASS CE1DY010_3"
              ELSE DISPLAY "-->FAIL CE1DY010_3"
           END-IF.
```

Figure 73. COBOL/370 Example illustrating calls to CEEDSHP, CEEGTST, and CEEDSHP

Figure 74 on page 285 contains a C/370 example in which all the LE/370 dynamic storage services are used. The example creates a new heap, then gets, changes the size, and frees a storage element from the new heap. The new heap is then discarded. The example also produces a storage report with a user-defined heading.

2. C/370 Example —

```
#pragma runopts(REPORT)
#include <leawi.h>
#include <string.h>

int main(void) {

   _CHAR80   heading;
   _FEEDBACK fc;

   /* initialize heading to blanks and then set the heading */
   memset(heading,' ',80);
   memcpy(heading,"User Defined Report Heading",27);

   /* set the report heading...do not need to check feedback token */
   /* because all return codes are successful                       */
   CEE3RPH(&heading,NULL);
   .
   .
   .
}
```

Examples Using Several Dynamic Storage Services

Figure 73 on page 284 contains a COBOL/370 example in which a new heap is defined with the same attributes as the user heap ("heap 0"), and storage is allocated from it. The heap is then discarded.

report_heading (input)
an 80-character fixed length string.

report_heading sets the identifying character string displayed at the top of the storage or options report. LE/370 uses only the first 79 bytes of *report_heading*; the last byte is ignored. *Report_heading* can contain DBCS characters surrounded by X'0E' (shift-out) and X'0F' (shift-in).

fc (output)
an optional 12-byte condition token returned by CEE3RPH indicating the result of the service.

If you specify *fc* when calling CEE3RPH, a mapping of the feedback condition code is returned to you.

The feedback codes from the service are as follows:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3JK	0	3700	The storage and options report heading replaced a previous heading.

Usage Notes

1. The character string specified in *report_heading* is printed on the storage report and the options report.
2. Figure 69 on page 242 contains a sample storage report produced by LE/370, and Figure 68 on page 240 contains a sample options report.

Examples

1. COBOL/370 Example —

```
77   RPTHEAD PIC X(80).
77   FC PIC X(12).
:

     MOVE "User defined report heading" TO RPTHEAD.
     CALL "CEE3RPH" USING RPTHEAD , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _INT4 heapid, size;
   _POINTER address;
   _FEEDBACK fc;
   #define SUCCESS "\0\0\0\0"
   .
   .
   .
   heapid = 0;     /* get storage from initial heap   */
   size = 4000;    /* number of bytes of heap storage */

   /* obtain the storage using CEEGTST */
   CEEGTST(&heapid,&size,&address,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEGTST failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }
   .
   .
   .
   /* free the storage that was previously obtained using CEEGTST */
   CEEFRST(&address,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEFRST failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }
   .
   .
   .
}
```

CEE3RPH — Set Report Heading

The CEE3RPH callable service allows you to set the heading displayed at the top of the storage or options reports that are generated when you specify the RPTSTG(ON) or RPTOPTS(ON) run-time options. Figure 69 on page 242 contains a sample storage report, and Figure 68 on page 240 contains a sample options report.

Syntax

```
►►──CEE3RPH──(──report_heading──,──┬────┬──)──────────────►◄
                                   └─fc─┘
```

Usage Notes

1. All requests for storage are conditional. If storage is not available, the feedback code (fc) is set and returned to you, but the thread does *not* ABEND. When storage is not available, the appropriate action in the member environment should be taken. One option is using the CEESGL callable service to signal the LE/370 condition handler with the returned feedback code. See "CEESGL — Signal a Condition" on page 311 for more information.

2. Storage obtained by CEEGTST can be freed by a call to CEEFRST or CEEDSHP. You can also free storage using a built-in language function. If storage is not explicitly freed, it is freed automatically at termination. For more information about CEEFRST and CEEDSHP, see "CEEFRST — Free Heap Storage" on page 276 and "CEEDSHP — Discard Heap" on page 274.

3. If you have specified a *heap_alloc_value* in the STORAGE run-time option, all storage allocated by CEEGTST is initialized to *heap_alloc_value*. Otherwise, it is left uninitialized. For more information about the STORAGE run-time option, see "STORAGE" on page 246.

4. If the value specified in the *size* parameter of CEEGTST is greater than the size of an increment (as specified in the HEAP run-time option), all of the requested storage (rounded up to the nearest doubleword) is allocated in a single system-level call.

5. Heap storage is acquired by a system-level get storage call in increments of *init_size* and *incr_size* bytes as specified by the HEAP run-time option, or in the CEECRHP call ("Create new additional heap"). If the increment size is chosen appropriately, only a few of the calls to CEEGTST result in a system call. The storage report generated when the RPTSTG run-time option is specified shows the number of system-level get storage calls required. This helps you tune the *init_size* and *incr_size* fields in order to minimize calls to the operating system. For more information about the HEAP and RPTSTG run-time options, see "HEAP" on page 230 and "RPTSTG" on page 240. For more information about the CEECRHP callable service, see "CEECRHP — Create New "Additional Heap"" on page 267.

Examples

1. COBOL/370 Example —

```
77   HEAPID PIC S9(9) COMP.
77   HPSIZE PIC S9(9) COMP.
77   ADDRSS PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE 0 TO HEAPID.
     MOVE 4000 TO HPSIZE.
     CALL "CEEGTST" USING HEAPID , HPSIZE , ADDRSS , FC.
```

heap_id (input)
a fullword signed integer.

heap_id is a token denoting the heap in which the storage is allocated. A *heap_id* of 0 allocates storage from the Initial Heap (or User Heap). Any other *heap_id* must be a value obtained from the CEECRHP (Create Heap) callable service. If the *heap_id* that you specify is invalid, no storage is allocated. CEEGTST terminates with a nonzero feedback code and the value of the *address* parameter is undefined.

For more information about the CEECRHP callable service, see "CEECRHP — Create New "Additional Heap"" on page 267.

size (input)
a fullword signed integer.

size represents the amount of storage to be allocated, in bytes. If the specified amount of storage cannot be obtained, no storage is allocated, CEEGTST terminates with a non-zero feedback code, and the value of the *address* parameter is undefined.

Note: In a CICS environment *size* should not exceed 1024 M (1 gigabyte or X'40000000') when running in AMODE ANY, and 65,504 bytes when running in AMODE(24). Portable applications should respect these CICS limitations.

address (output)
a fullword address pointer.

address is the machine address of the first byte of allocated storage. If storage cannot be obtained, *address* remains undefined. Storage is always allocated on a doubleword boundary.

Note: CEEGTST always allocates storage that is addressable by the caller. Therefore, if the caller is in AMODE(24), or if HEAP(,,BELOW) is in effect, the storage returned is always below-the-line. Above-the-line storage is returned only if the caller is in AMODE(31) and HEAP(,,ANY) is in effect. See "HEAP" on page 230 for further information about the LE/370 HEAP run-time option.

fc (output)
an optional 12-byte condition token indicating the result of the service.

If you specify *fc* as an argument when calling CEEGTST, information in the form of a condition token is returned to the calling routine.

The feedback codes from the service are as follows:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0P3	3	0803	The heap identifier in a get storage request or a discard heap request was unrecognized.
CEE0P8	3	0808	Storage size in a get storage request or a reallocate request was not a positive number.
CEE0PD	3	0813	Insufficient storage was available to satisfy a get storage request.

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _INT4 heapid, size;
   _POINTER address;
   _FEEDBACK fc;
   #define SUCCESS "\0\0\0\0"
   .
   .
   .
   heapid = 0;    /* get storage from initial heap   */
   size = 4000;   /* number of bytes of heap storage */

   /* obtain the storage using CEEGTST */
   CEEGTST(&heapid,&size,&address,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEGTST failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }

   .
   .
   .

   /* free the storage that was previously obtained using CEEGTST */
   CEEFRST(&address,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEFRST failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }
   .
   .
   .
}
```

CEEGTST — Get Heap Storage

The CEEGTST callable service allocates storage from a heap whose id you specify. It can be used to efficiently acquire both large and small blocks of storage.

Syntax

```
▸▸──CEEGTST──(──heap_id──,──size──,──address──,──┬────┬──)──▸◂
                                                 └─fc─┘
```

However, if you call CEEFRST for an invalid address, and you've specified TRAP(OFF), your application *may* ABEND. LE/370's reaction to this is undefined.

Also, partial freeing of an allocated area is not supported.

3. Storage allocated by CEEGTST, but not explicitly freed, is automatically freed at enclave termination.
4. CEEFRST generates a system-level free storage call to return a storage increment to the operating system only when:
   - The last heap element within an increment is being freed, *and*
   - either the HEAP run-time option or a call to CEECRHP specifies FREE (note that KEEP is the IBM-supplied default setting for the Initial Heap). See "HEAP" on page 230 for further information about the HEAP run-time option and "CEECRHP — Create New "Additional Heap"" on page 267 for information about CEECRHP.

   Otherwise, the freed storage is simply added to the free list; it is not returned to the operating system until termination. The out-of-storage condition can cause freeing of empty increments even when KEEP is specified.
5. If you specify *heap_free_value* in the STORAGE run-time option, all freed storage is overwritten with *heap_free_value*. Otherwise, it is simply marked "available."

   Note: Portions of the freed storage area may be used to hold internal storage manager control information. These areas will be overwritten, but not with *heap_free_value*. See "STORAGE" on page 246 for further information about the STORAGE run-time option.

Examples

1. COBOL/370 Example —

```
77   ADDRSS PIC S9(9) COMP.
77   FC PIC X(12).
:

     CALL "CEEFRST" USING ADDRSS , FC.
```

CEEFRST — Free Heap Storage

The CEEFRST callable service may be used to efficiently free storage previously allocated by CEEGTST or by a language built-in function. Normally, you do not need to call CEEFRST because the LE/370 heap manager automatically returns all heap storage to the operating system when the enclave terminates. However, if you are allocating a large amount of heap storage, you should free the storage when it is no longer needed. This freed storage then becomes available to satisfy later requests for heap storage, thus reducing the total amount of storage needed to run the application.

Syntax

```
►►──CEEFRST──(──address──,──┬────┬──)──────────────────────────►◄
                            └─fc─┘
```

address (input)

a fullword address pointer.

address is the address of the storage that is freed. It is the same address returned by a previous CEEGTST call.

If the storage is freed successfully, the value of *address* becomes undefined. If unsuccessful, CEEFRST sets a nonzero feedback code; *address* is left unchanged.

fc (output)

an optional 12-byte condition token returned by CEEFRST indicating the result of the service.

If you specify *fc* as an argument when calling CEEFRST, feedback information in the form of a condition token is returned to the calling routine.

The feedback codes from the service are as follows:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0PA	3	0810	The storage address in a free storage request was not recognized, or heap storage control information was damaged.

Usage Notes

1. All requests to free storage are conditional. If storage cannot be freed, the appropriate feedback code is set but, if possible, the application does not ABEND.
2. An attempt to free storage that was already marked as free produces no action and returns a nonzero feedback code (fc). An attempt to free storage at anything other than a valid starting address produces no action and returns a nonzero feedback code.

Examples

1. COBOL/370 Example —

```
77   HEAPID PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE 0 TO HEAPID.
     CALL "CEEDSHP" USING HEAPID , FC.
```

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _INT4 heapid, size, increment, options;
   _FEEDBACK fc;
   #define SUCCESS "\0\0\0\0"
   .
   .
   .
   heapid = 0;         /* heap identifier will be set by CEECRHP */
   size = 4096;        /* initial size of heap  (in bytes)       */
   increment = 4096;   /* increment to extend the heap by        */
   options = 71;       /* set up heap as (,,ANYWHERE,FREE)       */

   /* create heap using CEECRHP */
   CEECRHP(&heapid,&size,&increment,&options,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEECRHP failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }

   .
   .
   .
   /* discard the heap that was previously created using CEECRHP */
   CEEDSHP(&heapid,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEDSHP failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }
   .
   .
   .
}
```

CEEDSHP — Discard Heap

The CEEDSHP callable service allows you to discard an entire heap that you created previously with a call to CEECRHP. CEECRHP returns a unique *heap_id* to the caller; use this id in the call to CEEDSHP. A *heap_id* of 0 is not permitted with CEEDSHP.

Syntax

```
▸▸——CEEDSHP—(—heap_id—,—┬────┬—)————————————————————————▸◂
                        └─fc─┘
```

heap_id (input)

a fullword signed integer.

heap_id is a token specifying the heap to be discarded.

fc (output)

an optional 12-byte condition token returned by CEEDSHP indicating the result of the service.

If you specify *fc* as an argument when calling CEEDSHP, feedback information in the form of a condition token is returned to the calling routine.

The feedback codes from the service are as follows:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0P3	3	0803	The heap identifier in a get storage request or a discard heap request was unrecognized.
CEE0PC	3	0812	An invalid attempt to discard the Initial Heap was made.

Usage Notes

1. A *heap_id* of 0 is invalid; the Initial Heap is logically created during enclave initialization and may not be discarded.
2. After the call to CEEDSHP, any existing pointers to storage which have been allocated from this heap are "dangling" pointers, that is, pointers to storage that has been freed. Use of these pointers can cause unpredictable, probably erroneous, results.
3. Discarding a heap with CEEDSHP immediately returns all storage allocated to the heap to the operating system, even if the KEEP suboption had been specified with the HEAP run-time option.

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _INT4 heapid, size;
   _POINTER address;
   _FEEDBACK fc;
   #define SUCCESS "\0\0\0\0"
   .
   .
   .
   heapid = 0;     /* get storage from initial heap    */
   size = 4000;    /* number of bytes of heap storage */

   /* obtain the storage using CEEGTST */
   CEEGTST(&heapid,&size,&address,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEGTST failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }

   .
   .
   .
   size = 2000;  /* new size of storage element */

   /* change the size of the storage element */
   CEECZST(&address,&size,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEECZST failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }
   .
   .
   .
   /* free the storage that was previously obtained using CEEGTST */
   CEEFRST(&address,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEFRST failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }
   .
   .
   .
}
```

3. The new heap element is allocated from the same heap that contained the old storage. The *address* specified in CEECZST indicates to LE/370 the heap to which the storage belongs.

Examples

1. COBOL/370 Example —

```
77   ADDRSS PIC S9(9) COMP.
77   NEWSIZE PIC S9(9) COMP.
77   FC PIC X(12).
⋮

     MOVE 2000 TO NEWSIZE.
     CALL "CEECZST" USING ADDRSS , NEWSIZE , FC.
```

Syntax

```
▸▸──CEECZST──(──address──,──new_size──,─┬────┬─)──────────────▸◂
                                        └─fc─┘
```

address (input/output)
a fullword address pointer.

On input, this parameter contains an address returned by a previous CEEGTST call. On output, the address of the first byte of the newly allocated storage is returned in this parameter.

new_size (input)
a fullword signed integer.

new_size is the number of bytes of storage to be allocated for the new heap element.

fc (output)
an optional 12-byte condition token returned by CEECZST indicating the result of the service.

If you specify *fc* as an argument in a call to CEECZST, feedback information in the form of a condition token is returned to the calling routine. For more information on data type definitions, see "Data Type Definitions" on page 262.

The feedback codes from the service are as follows:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0P8	3	0808	Storage size in a get storage request or a reallocate request was not a positive number.
CEE0PA	3	0810	The storage address in a free storage request was not recognized, or heap storage control information was damaged.
CEE0PD	3	0813	Insufficient storage was available to satisfy a get storage request.

Usage Notes

1. The contents of the old storage are preserved in the following manner:
   - If *new_size* is equal to or greater than the old size, the entire contents of the old element are copied to the new element. The remaining bytes in the new element are left uninitialized (unless the STORAGE run-time option is in effect — see "STORAGE" on page 246 for more information).
   - If *new_size* is less than the old size, the contents of the old element are truncated to the size of the new element.

   Note: The address of the old element can also be changed.
2. Because the new storage can be allocated at a different location from the existing allocation, any pointers referring to the old storage are invalid.

2. C/370 Example —

```
#include <leawi.h>
#include <stdio.h>
#include <string.h>

int main(void) {

   _INT4 heapid, size, increment, options;
   _FEEDBACK fc;
   #define SUCCESS "\0\0\0\0"
   .
   .
   .
   heapid = 0;         /* heap identifier will be set by CEECRHP */
   size = 4096;        /* initial size of heap  (in bytes)       */
   increment = 4096;   /* increment to extend heap by            */
   options = 71;       /* set up heap as (,,ANYWHERE,FREE)       */

   /* create heap using CEECRHP */
   CEECRHP(&heapid,&size,&increment,&options,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEECRHP failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }
   .
   .
   .
   /* discard the heap that was previously created using CEECRHP */
   CEEDSHP(&heapid,&fc);

   /* check the first 4 bytes of the feedback token (0 if successful) */
   if (memcmp(&fc,SUCCESS,4) != 0) {
      printf("CEEDSHP failed with message number %d\n",fc.tok_msgno);
      exit(99);
   }
   .
   .
   .
}
```

CEECZST — Reallocate (Change siZe of) STorage

The CEECZST callable service allows you to change the size of a previously allocated heap element. The *address* parameter points to the beginning of the heap element. The *new_size* parameter gives the new size of the heap element, in bytes. The contents of the heap element are unchanged up to the shorter of the new and old sizes.

The CEECZST service returns a pointer to the reallocated heap element. The storage location of the heap element might be moved by the CEECZST service. Thus, the *address* parameter passed to CEECZST is not necessarily the same as the value returned.

Examples

1. COBOL/370 Example —

```
77   HEAPID PIC S9(9) COMP.
77   HPSIZE PIC S9(9) COMP.
77   INCR PIC S9(9) COMP.
77   OPTS PIC S9(9) COMP.
77   FC PIC X(12).
:

     MOVE 0 TO HEAPID.
     MOVE 1 TO HPSIZE.
     MOVE 0 TO INCR.
     MOVE 0 TO OPTS.
     CALL "CEECRHP" USING HEAPID , HPSIZE , INCR , OPTS , FC.
```

Table 46. HEAP Attributes Based on Setting of the Option Parameter

Option Setting	HEAP attributes
00	Use same attributes as the Initial Heap (copy them from the HEAP run-time option)
01	HEAP(,,,FREE) (location inherited from HEAP run-time option)
70	HEAP(,,,KEEP) (location inherited from HEAP run-time option)
71	HEAP(,,ANYWHERE,KEEP)
72	HEAP(,,ANYWHERE,FREE)
73	HEAP(,,BELOW,KEEP)
74	HEAP(,,BELOW,FREE)
75	HEAP(,,ANYWHERE,)(disposition inherited from the HEAP run-time option)
76	HEAP(,,BELOW,)(disposition inherited from the HEAP run-time option)

fc (output)

an optional 12-byte *condition token* (data type FEEDBACK) returned by CEECRHP indicating the result of the service.

If you specify *fc* as an argument when calling CEECRHP, feedback information in the form of a condition token is returned to the calling routine.

The feedback codes from the CEECRHP callable service are as follows:

Symbolic LE FBCode	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0P4	3	0804	The initial size value supplied in a create heap request was unsupported.
CEE0P5	3	0805	The increment size value supplied in a create heap request was unsupported.
CEE0P6	3	0806	The options value supplied in a create heap request was unrecognized.
CEE0PD	3	0813	Insufficient storage was available to satisfy a get storage request.

Usage Notes

1. The counterpart of CEECRHP, which creates a new "Additional Heap", is CEEDSHP, which discards an entire heap. See "CEEDSHP" below for more information.
2. The number of heaps supported by LE/370 is limited only by the amount of virtual storage available.
3. The allocation of heap storage for the new heap is deferred until the first CEEGTST call that uses the new *heap_id*. CEECRHP obtains only a small amount of storage from a private LE/370 heap to keep track of the newly created heap.

Chapter 34. Dynamic Storage Callable Services

CEECRHP — Create New "Additional Heap"

The CEECRHP callable service allows you to define additional heaps. It returns a unique *heap_id*. The heaps defined by CEECRHP can be used just like the LE/370 Initial Heap (HEAPID=0). Unlike the Initial Heap, however, *all* heap elements within an additional heap can be quickly freed by using a single call to CEEDSHP (DiScard HeaP).

Syntax

```
▸▸——CEECRHP—(—heap_id—,—initial_size—,—increment—,——————▸

▸——options—,—┬——┬—)——————————————————————————————▸◂
             └fc┘
```

heap_id (output)
a fullword signed integer.

heap_id is the heap identifier of the created heap. If a new heap cannot be created, the value of *heap_id* remains undefined.

initial_size (input)
a fullword signed integer.

initial_size is the initial amount of storage, in bytes, that is to be allocated for the new heap. *initial_size* is rounded up to the nearest increment of 4096 bytes.

If *initial_size* is specified as 0, then the *init_size* specified in the HEAP run-time option is used. If no HEAP run-time option was provided and *initial_size* is specified as 0, CEECRHP uses the installation default. See "HEAP" on page 230 for more information about the HEAP run-time option and IBM-supplied defaults.

increment (input)
a fullword signed integer.

When it is necessary to enlarge the heap to satisfy an allocation request, *increment* represents the number of bytes by which the heap will be extended. If *increment* is specified as 0, then the *incr_size* specified in the HEAP run time option is used. If no HEAP run-time option was provided and *increment* equals 0, CEECRHP uses the installation default. See "HEAP" on page 230 for more information about the HEAP run-time option and IBM-supplied defaults.

options (input)
a fullword signed integer.

Options are specified with the decimal codes as shown in Table 46 on page 268.

© Copyright IBM Corp. 1991

Table 45 (Page 4 of 4). Callable Services Quick Reference

Callable Service	Function	Page
▸▸—CEE3CNC—(—allow—,—┬────┬—)—▸◂ └─fc─┘	Allows or prohibits nested conditions within a condition handler that you registered using the CEEHDLR callable service.	315
▸▸—CEE3CTY—(—function—,—country_code—,—┬────┬—)—▸◂ └─fc─┘	Allows the calling routine to change or query the current national country setting.	345
▸▸—CEE3DMP—(—title—,—options—,—┬────┬—)—▸◂ └─fc─┘	Generates a dump of the run-time environment of LE/370 and the member language libraries.	421
▸▸—CEE3GRN—(—name—,—┬────┬—)—▸◂ └─fc─┘	Obtains the name of the most current LE/370-enabled program.	316
▸▸—CEE3LNG—(—function—,—desired_language—,—┬────┬—)—▸◂ └─fc─┘	Allows the calling routine to change or query the current national language.	348
▸▸—CEE3MCS—(—country_code—,—currency_symbol—,—┬────┬—)—▸◂ └─fc─┘	Returns the default currency symbol for the country you specify in *country_code*.	352
▸▸—CEE3MTS—(—country_code—,—thousands_separator—,—▸◂ ▸—┬────┬—)—▸◂ └─fc─┘	Returns the default thousands separator for the country that you specify in *country_code*	354
▸▸—CEE3PRM—(—char_parm_string—,—┬────┬—)—▸◂ └─fc─┘	Returns to the calling routine the parameter string that was specified at invocation of the program.	416
▸▸—CEE3RPH—(—report_heading—,—┬────┬—)—▸◂ └─fc─┘	Allows you to set the heading displayed at the top of the storage or options reports that are generated when you specify the RPTSTG(ON) or RPTOPTS(ON) run-time options.	281
▸▸—CEE3SPM—(—action—,—cond_string—,—┬────┬—)—▸◂ └─fc─┘	Used to query and modify the enablement of LE/370 hardware conditions.	318
▸▸—CEE3USR—(—function_code—,—field_number—,—field_value—,—▸ ▸—┬────┬—)—▸◂ └─fc─┘	Sets or queries one of two 4-byte fields known as the user area fields.	417

Table 45 (Page 3 of 4). Callable Services Quick Reference

Callable Service	Function	Page
►►—CEEITOK—(—I_CTOK—,—[fc]—)—►◄	Returns the initial condition token for the current "Condition Information Block."	298
►►—CEELOCT—(—output_Lilian—,—output_seconds—,—► ►—output_Gregorian—,—[fc]—)—►◄	Returns the current Local Time in three formats.	383
►►—CEEMGET—(—Cond_Token—,—Message_Area—,—Msg_Index—► ►—[fc]—)—►◄	Retrieves, formats and stores in a buffer a message corresponding to a condition token either returned for a callable service or passed to a user-written condition handler.	327
►►—CEEMOUT—(—Message_String—,—Destination_Code—,—► ►—[fc]—)—►◄	Dispatches a user-defined message to the message file.	330
►►—CEEMRCR—(—type_of_move—,—[fc]—)—►◄	Moves the resume cursor to a position relative to the current position of the handle cursor.	300
►►—CEEMSG—(—Cond_Token—,—Destination_Code—,—[fc]—)—►◄	Is designed to obtain /format /dispatch a message corresponding to an input condition token received from a callable service or passed to a user-written condition handler.	331
►►—CEENCOD—(—C_1—,—C_2—,—Case—,—Severity—,—Control—,—► ►—Facility_ID—,—I_S_Info—,—Cond_Token—,—[fc]—)—►◄	Used to dynamically construct a condition token.	306
►►—CEEQCEN—(—century_start—,—[fc]—)—►◄	Queries the century within which LE/370 assumes 2-digit year values lie.	385
►►—CEERAN0—(—seed—,—random_no—,—[fc]—)—►◄	Generates a sequence of uniform pseudo-random numbers between 0 and 1 using the multiplicative congruential method with a user-specified seed.	426
►►—CEESCEN—(—century_start—,—[fc]—)—►◄	Sets the century in which LE/370 assumes 2-digit year values lie.	387
►►—CEESECI—(—input_seconds—,—output_year—,—output_month—,—► ►—output_day—,—output_hours—,—output_minutes—,—► ►—output_seconds—,—output_milliseconds—,—[fc]—►◄	Converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond.	389
►►—CEESECS—(—input_timestamp—,—picture_string—,—► ►—output_seconds—,—[fc]—)—►◄	Converts a string representing a timestamp into a number representing the number of seconds since 00:00:00 14 October 1582.	391
►►—CEESGL—(—cond—,—[q_data_token,]—[fc]—)—►◄	Raises, or signals, a condition to the condition manager.	311
►►—CEETEST—(—[string_of_commands]—,—[fc]—)—►◄	Invokes a debug tool, such as AD/Cycle CODE/370.	419
►►—CEE3ABD—(—abcode—,—timing—)—►◄	Terminates the enclave immediately with an ABEND.	313

Table 45 (Page 2 of 4). Callable Services Quick Reference

Callable Service	Function	Page
▸▸—CEEDSHP—(—heap_id—,—┬────┬—)—▸◂ └fc┘	Allows you to discard an entire heap that you created previously with a call to CEECRHP.	274
▸▸—CEEDYWK—(—input_Lilian_date—,—output_day_no—,—┬────┬—)—▸◂ └fc┘	Calculates the day of the week on which a Lilian date falls.	374
▸▸—CEEFRST—(—address—,—┬────┬—)—▸◂ └fc┘	Can be used to free storage previously allocated by CEEGTST or a language intrinsic function.	276
▸▸—CEEFMDA—(—country_code—,—date_pic_str—,—┬────┬—)—▸◂ └fc┘	Returns to the calling routine the default date picture string for a specified country.	
▸▸—CEEFMDS—(—country_code—,—decimal_separator—,—▸◂ ▸—┬────┬—▸◂ └fc┘	Returns the default decimal separator for the country specified in the country_code.	339
▸▸—CEEFMDT—(—country_code—,—datetime_str—,—┬────┬—)—▸◂ └fc┘	Returns the default date and time picture strings for the country specified in the *country_code*.	341
▸▸—CEEFMTM—(—country_code—,—time_pic_str—,—┬────┬—)—▸◂ └fc┘	The CEEFMTM callable service returns to the calling routine the default time picture string for the country specified in the *country_code*.	343
▸▸—CEEGMT—(—output_GMT_Lilian—,—output_GMT_seconds—,—▸ ▸—┬────┬—)—▸◂ └fc┘	Returns the current Greenwich Mean Time (GMT) as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582.	376
▸▸—CEEGMTO—(—offset_hours—,—offset_minutes—,—▸ ▸—offset_seconds—,—┬────┬—)—▸◂ └fc┘	Returns values to the calling routine that represent the difference between the local system time and Greenwich Mean Time.	378
▸▸—CEEGPID—(—CEE_Version_ID—,—Plat_ID—,—┬────┬—)—▸◂ └fc┘	Retrieves the LE/370 version ID and the platform ID of the platform and version of LE/370 that is currently in use for processing the currently active condition.	290
▸▸—CEEGQDT—(—cond_rep—,—q_data_token—,—┬────┬—)—▸◂ └fc┘	Provides a mechanism by which application code (in particular, user condition handlers) can retrieve the *q_data_token* from the Instance Specific Information(ISI).	291
▸▸—CEEGTST—(—heap_id—,—size—,—address—,—┬────┬—)—▸◂ └fc┘	Allocates storage from a heap whose id you specify.	278
▸▸—CEEHDLR—(—routine—,—token—,—┬────┬—)—▸◂ └fc┘	Used to register a user condition handler for the current stack frame. "debugging lines" or the USE FOR DEBUGGING declarative.	294
▸▸—CEEHDLU—(—routine—,—┬────┬—)—▸◂ └fc┘	Unregisters a user condition handler for the current stack frame.	296
▸▸—CEEISEC—(—input_year—,—input_month—,—input_day—,—▸ ▸—input_hours—,—input_minutes—,—input_seconds—,—▸ ▸—input_milliseconds—,—output_seconds—,—┬────┬—)—▸◂ └fc┘	Converts separate binary integers representing year, month, day, hour, minute, second, and millisecond to a number representing the number of seconds since 00:00:00 14 October 1582.	380

Data Type	Description	COBOL/370:	C/370:
INT2	A 2-byte signed integer	PIC S9(4)	signed short
INT4	A 4-byte signed integer	PIC S9(9)	signed int
FLOAT4	A 4-byte single-precision floating-point number	COMP-1	float
FLOAT8	An 8-byte double-precision floating-point number	COMP-2	double
POINTER	A platform-dependent address pointer	USAGE IS POINTER	void *
CHAR*n*	A string (character array) of unknown length	PIC X(n)	char[n]

VSTRING	A string of arbitrary length used for polymorphic string parameter declarations. The string may be any one of a fixed-length string, a null-terminated varying string (known as an "ASCIIZ") or a length-prefixed string.	`01 string`
		`  02 len pic 9(9) binary`
		`  02 txt pic x(n)`
			`struct {`
			`_INT2   length;`
			`char   string:1";`
			`}`

FEED_BACK	A mapping of the condition token (fc)	`01 fc`
		`  02 sev   pic x(2)`
		`  02 msgno pic x(2)`
		`  02 flgs  pic x(1)`
		`  02 facis pic x(3)`
		`  02 isi   pic x(4)`
			`typedef struct {`
			`   short    tok_sev    ;`
			`   short    tok_msgno  ;`
			`   int      tok_case :2,`
			`            tok_sever:3,`
			`            tok_ctrl :3 ;`
			`   char     tok_facid[3];`
			`   int      tok_isi    ;`
			`        }          _FEEDBACK;`

CEE_ENTRY	A platform- and HLL-dependent entry constant	PROCEDURE_POINTER	`struct {`
			`_POINTER address;`
			`_POINTER nesting;`
			`}`

Table 44. Data Type Definitions across LE/370-enabled HLLs

Quick Reference of LE/370 Callable Services

Table 45 (Page 1 of 4). Callable Services Quick Reference

Callable Service	Function	Page

`▸▸──CEECRHP──(──heap_id──,──initial_size──,──increment──,──▸`
`▸──options──,──┬────┬──)──▸◂`
`               └─fc─┘`
	Allows you to define additional heaps.	267

`▸▸──CEECZST──(──address──,──new_size──,──┬────┬──)──▸◂` `                                         └─fc─┘`	Allows you to change the size of a previously allocated storage element, thus preserving its contents.	270

`▸▸──CEEDATE──(──input_Lilian_date──,──picture_string──,──▸`
`▸──output_char_date──,──┬────┬──)──▸◂`
`                        └─fc─┘`
	Converts a number representing a Lilian date to a date written in character format.	368

`▸▸──CEEDATM──(──input_seconds──,──picture_string──,──▸`
`▸──output_timestamp──,──┬────┬──)──▸◂`
`                       └─fc─┘`
	Converts a number representing the number of seconds since 00:00:00 14 October 1582 to character format.	371

`▸▸──CEEDAYS──(──input_char_date──,──picture_string──,──▸`
`▸──output_Lilian_date──,──┬────┬──)──▸◂`
`                         └─fc─┘`
	Converts a string representing a date to a Lilian format.	361

`▸▸──CEEDCOD──(──Cond_Token──,──C_1──,──C_2──,──Case──,──▸`
`▸──Severity──,──Control──,──Facility_ID──,──I_S_Info──,──▸`
`▸──┬────┬──▸◂`
`    └─fc─┘`
	Allows you to decompose or alter an existing condition token.	287

If specified as an argument, feedback information in the form of a condition token is returned to the calling routine. The condition token will indicate whether the service completed successfully, or if a condition was encountered during execution. LE/370 provides you with the ability to decode the condition token so that it can be acted on.

If *fc* is omitted as an argument, the condition will be signaled if the service was not successful. See Chapter 18, "Communicating Conditions" on page 79 for more information about condition tokens.

Usage Notes

1. C Considerations — You must invoke LE/370 callable services as *procedures.* You cannot invoke any callable service as a *function.*

   In C/370, you omit a parameter by passing a null pointer to the callable service in place of the parameter, as indicated in Figure 72 on page 261.

   Note that input strings for callable services are *not* NULL terminated in C/370.

2. COBOL Considerations — You can call LE/370 services either statically or dynamically.

3. You can invoke callable services from any LE/370-enabled HLL except where otherwise noted. Assembler applications can also invoke callable services by adhering to the requirements described in "Invoking Callable Services from Assembler Routines" on page 170.

4. Most callable services are available on any platform that LE/370 supports (this is restricted to System/370 in Language Environment/370 Version 1). Callable services that are available only on System/370 in Language Environment/370 Version 1 begin with the prefix **CEE3**. Callable services provided by LE/370 that are cross-system consistent begin with the prefix **CEE** only.

5. Routines that invoke callable services do not need to be AMODE(31). AMODE switching is performed implicitly without any action required by the calling routine, if you specify the ALL31(OFF) run-time option (see "ALL31" on page 218).

6. Under LE/370, all parms are passed by reference, indirectly.

Data Type Definitions

Parameters in LE/370 are defined as specific data types, for example:

- Fullword binary integer
- Short floating point hex
- Long floating point hex
- A fixed length character string with a pre-defined length
- An entry variable
- A character string with a halfword prefix indicating its current length.

The data type definitions that are used in the callable service syntax descriptions in this book are described in Table 44 on page 263.

Chapter 33. Invoking Callable Services

LE/370 callable services can be invoked by Assembler routines, HLL-generated object code, HLL library routines, other LE/370 library routines, and user-written HLL calls. User-written HLL calls and functions access LE/370 library routines using the same mechanisms currently used by HLLs to support calls in general.

If you are a COBOL/370 user, you can invoke LE/370 callable services using the syntax shown in Figure 70.

```
CALL      CEESERV USING parm1, parm2, ... parmn, fc
```

Figure 70. Sample Callable Services Invocation Syntax for COBOL/370 Users

If you are a C user, you can invoke LE/370 callable services using the syntax shown in Figure 71.

```
#include <leawi.h>
int main(void)
{
 CEESERV(parm1, parm2, ... parmn, fc);
 }
```

Figure 71. Sample Callable Services Invocation Syntax for C/370 Users

```
#include <leawi.h>
int main(void)
{
 CEESERV(parm1, parm2, ... parmn, OMIT_FC);
}                                  /* OMIT_FC is defined */
                                  /* in the leawi.h header file */
```

Figure 72. Example of Omitted Parameter. The figure illustrates how to omit the fc parameter in a call to an LE/370 service. Only C/370 routines may omit parameters in the call.

where:

CEESERV
is the name of the callable service

parm1 parm2 ... parm*n*
are optional or required parameters passed to or returned from the called service.

Note that some callable service parameters are optional, but for C/370 users only. If you do not want to pass the parm or you do not want the return value, pass NULL instead of the parm, as indicated in Figure 72.

fc
is an optional feedback code that indicates the result of the service. *fc* can be omitted by HLLs, such as C, that support optional parameters.

© Copyright IBM Corp. 1991

IBM SAA AD/Cycle Language Environment/370

SC26-4818-00

Programming Guide

Version 1 Release 1

Dave Knudson
E222

IBM

IBM SAA AD/Cycle Language Environment/370

SC26-4818-00

Programming Guide

Version 1 Release 1

Comments

161-167 ✓
225 ✓
174, 177
339-45?? ✓

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page ix.

First Edition (December, 1991)

This edition applies to Version 1 Release 1 of IBM SAA AD/Cycle Language Environment/370, Program Number 5688-198, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to this publication; make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department J58
P. O. Box 49023,
San Jose, California, U.S.A. 95161-9023

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1991. All rights reserved.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices ix
Programming Interface ix
Trademarks ix

About this Book x
LE/370's Position in the AD/Cycle Framework x
Programming Tasks and the Publications Library Structure xiii
Publications Provided with LE/370 xiii
How to Read the Syntax Diagrams xiv

Introduction to LE/370 1

Chapter 1. LE/370 Overview 2
What is LE/370? 2
Benefits of a Common Run-time Environment 2
Compatibility 3

Chapter 2. Program Model 4
Program Model Description 4
Terminology 5
Summary of Resource Ownership 8

Considerations for Writing an LE/370 Application 9

Chapter 3. Parameter List Formats 10
Argument Lists and Parameter Lists 10
Methods for Passing Arguments to and from Routines 10
Format of Arguments Passed by Operating Systems and Subsystems 12
C/370 Parameter Passing Styles 14

Chapter 4. Application Return Codes 20
Application Terminating Events 20
Enclave Return Code Processing 21

Chapter 5. Run-time User Exits 23

Linking and Running Your Program 25

Chapter 6. LE/370 Library Routine Considerations 27

Chapter 7. Making Your Application Reentrant 28
Making Your COBOL Routine Reentrant 29
Making Your C/370 Program Reentrant 29

Chapter 8. Link-editing with LE/370 30
Link-editing Single-Language Applications 30
Link-editing ILC Applications 30

© Copyright IBM Corp. 1991

Chapter 9. Using the Linkage-Editor Under MVS 31
Providing Input to the Linkage-Editor 31
Linkage Editor Options 35

Chapter 10. Using the Loader under MVS 37
Providing Input to the Loader 37

Chapter 11. Running Your Application under MVS 41
Using the EXEC Statement 41
Specifying Run-time Options 42

Chapter 12. Creating a Load Module under TSO 44
Link-editing Your Application Using the LINK Command 44
Loading and Running a Load Module Using the LOADGO Command 45

Chapter 13. Running Your Application under TSO 48
Using the CALL Command 48

Chapter 14. Link-editing under VM/CMS 50
Using the GLOBAL Command 50
Search Order for Dynamic Routines 51
Using the LOAD Command 51
Using the GENMOD command 53
Using the LKED Command 54

Chapter 15. Running Your Application under CMS 56
Using the START command 56
Using the C/370 CMOD EXEC 59
Using the C/370 LINKLOAD EXEC 61

Chapter 16. Using IBM-supplied Cataloged Procedures 62
Invoking Cataloged Procedures 62
IBM-supplied Cataloged Procedures 63
Modifying Cataloged Procedures 70

Managing the Language Environment 73

Chapter 17. Stack and Heap Storage 74
Stack Storage 74
Heap Storage 76

Chapter 18. Communicating Conditions 79
Condition Token Layout 79
Relationship of the Condition Token to Messages 80

Chapter 19. Condition Management 83
Sources of Conditions 83
Obtaining the Stack Frame 83
Handlers Invoked During Condition Handling 84
Responses to Conditions 85
Condition Management Model 86
ERRCOUNT Run-time Option 90
Language-specific Condition Handling Semantics 90

Nested Conditions 97
LE/370-issued ABENDs 97
Using XUFLOW and CEE3SPM to Enable and Disable Hardware Conditions . 98
Callable Services Available to Handle Conditions 100
Examples 100

Chapter 20. Message Handling 110
Delivering the Message 110
Dynamic Allocation of the MSGFILE ddname 110
Inserting Messages in Your Application 111
Message Handling and National Language Support 115
Using COBOL/370 DISPLAY and ACCEPT statements 116

Subsystem Considerations 119

Chapter 21. CICS Considerations 120
Mapping the CICS Program Model to the LE/370 Program Model 120
Developing an Application under CICS 121
ILC under CICS 124
Support for Calls within Same HLL under CICS 125
Storage Management 126
Condition Handling 126
Run-time Output 129

Chapter 22. IMS Considerations 131
Using the Interface between LE/370 and IMS 131
IMS Communication with Your Application 132
Storage Considerations 133
Condition Handling under IMS 133
Making Your Application Reentrant 134

Chapter 23. DB2 Considerations 135
Overview of DB2 Processing 135

Interlanguage Communication (ILC) 137

Chapter 24. ILC Overview 138
Special Considerations 138

Chapter 25. C/370—COBOL Interlanguage Communication 140
LE/370 Interlanguage Communication Support 140
Compatibility with Pre LE/370-conforming Applications 140
C/370—COBOL/370 Calls Permitted under LE/370 140
C/370 — COBOL/370 Special Declarations 141
General Considerations for Static and Dynamic ILC Calls 142
Sample ILC Application 155

Advanced Topics 157

Chapter 26. Assembler Considerations 158
Register Conventions 158

General Considerations 159
Using Assembler Macros 160
Invoking Callable Services from Assembler Routines 170
System Services Available to Assembler Routines 170

Chapter 27. Advanced User Exit Topics 173
Using the Assembler User Exit to Tailor the Environment 173
High-Level Language User Exit Interface 183

Chapter 28. Pre-initialization 185
Using the CEEPIPI Pre-initialized Interface 185
Using the PIPI Table 186
Reentrancy Considerations 188
User Exit Invocation 189
Stop Semantics 189
CEEPIPI Syntax 190
Service Routines 198

Chapter 29. Nested Enclaves 202
Additional Nested Enclave Considerations 202

Using Run-time Options 205

Chapter 30. Specifying Run-time Options 206
Order of Precedence 207
Specifying Run-time Options and Program Arguments 207

Chapter 31. Run-time Options 213
Quick Reference of LE/370 Run-time Options 213
LE/370 Run-time Options 216

Chapter 32. Language Run-time Option Mapping 255

Using Callable Services 259

Chapter 33. Invoking Callable Services 261
Data Type Definitions 262
Quick Reference of LE/370 Callable Services 263

Chapter 34. Dynamic Storage Callable Services 267

Chapter 35. Condition Handling 287

Chapter 36. Message Handling 327

Chapter 37. National Language Support 338

Chapter 38. Date and Time Services 359

Chapter 39. Math Routines 406
Calling the Math Routines 406
Math Routine Descriptions 407

Feedback Code Descriptions 411
COBOL Considerations 413
Examples 413

Chapter 40. General LE/370 Callable Services 416

Chapter 41. Guidelines for Writing an LE/370 Callable Service 429

Appendix A. Pre-linking Your C Application 431
C/370 Prelinkage Utility Input Processing 431
C/370 Prelinkage Utility Automatic Call Library Processing 432
C/370 Prelinkage Utility Mapping of L-names to S-names 433
Invoking the Prelink Facility under MVS Batch 435
Invoking the Prelink Facility under TSO 436
Prelink Options 437
C/370 Prelinkage Utility Listing 438
Control Statement Processing 441

Appendix B. Building Applications in Systems Programming Environment 445
Building Freestanding Applications 445
Building Freestanding Applications under VM 446
Building Freestanding Applications under MVS 448
Building System Exit Routines 451
Building Persistent C Environments 451
Building User-Server Environments 451
Summary 451

Appendix C. Using the C/370 Object Library Utility 453
Creating an Object Library under VM 453
Creating an Object Library under MVS Batch 455
Creating an Object Library under TSO 456
C/370 Object Library Utility Map 457

Appendix D. Using the C/370 Multitasking Facility 461
Compiling and Linking Applications That Use MTF 461
Running Applications That Use MTF 463

Appendix E. Sort/Merge Considerations 465
Condition Handling Considerations 466

Appendix F. Running COBOL programs under ISPF 468

Appendix G. IBM-supplied Country Code Defaults 469

Appendix H. LE/370 Macros 476

Bibliography 477
LE/370 Publications 477
High Level Language Publications 477
Related Publications 477

LE/370 Glossary 479

Index 488

Notices

References in this publication to IBM products or services do not imply that they will be available everywhere IBM operates, nor that only IBM products or services can be used. Functionally equivalent products or services that do not infringe IBM's legal rights can be used instead. Operation with products or services other than those expressly designated by IBM is your responsibility.

IBM may have patents or pending patent applications covering subject matter described herein. This document neither grants nor implies any license or immunity under any IBM or third-party patents, patent applications, trademarks, copyrights, or other similar rights, or any right to refer to IBM in any marketing activities. Other than responsibilities assumed via the Agreement for Purchase of IBM Machines and the Agreement for IBM Licensed Programs, IBM assumes no responsibility for any infringement of third-party rights that may result from use of the subject matter disclosed in this publication or from the manufacture, use, lease, or sale of machines or programs described herein.

Licenses under IBM's utility patents are available on reasonable and nondiscriminatory terms. IBM does not grant licenses under its appearance design patents. Direct licensing inquiries in writing to the IBM Director of Commercial Relations, International Business Machines Corporation, Purchase, New York 10577.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. **This disclaimer does not apply in the United Kingdom or elsewhere if inconsistent with the local law.**

Programming Interface

This book is intended to help with application programming. This book documents General-Use Programming Interface and Associated Guidance Information provided by IBM SAA AD/Cycle Language Environment/370.

General-Use programming interfaces allow the customer to write programs that obtain the services of IBM SAA AD/Cycle Language Environment/370.

Trademarks

The following terms, denoted by an asterisk (*) on their first occurrences in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

AD/Cycle	ESA	RPG
AS/400	IBM	SAA
CICS/ESA	IMS/ESA	SQL
Cross System	KnowledgeTool	Systems Application Architecture
CUA	MVS/ESA	System/370
DB2	OS/2	VM/ESA

© Copyright IBM Corp. 1991

About this Book

This book is intended to help you run applications under LE/370 that are written in LE/370-conforming High Level Languages and Assembler. Previous versions of these language products provided their own environment and services for running applications, and their associated Application Programming Guides commonly included information on how to link-edit and execute applications. LE/370 now provides the run-time environment and services required to run applications compiled under all supported language products. This book therefore contains information about linking, running, and using services within the LE/370 environment applicable to all LE/370-conforming languages, as well as language-specific and operating system information when necessary.

For a review of supported High Level Language and LE/370 publications, see "Programming Tasks and the Publications Library Structure" on page xiii.

LE/370's Position in the AD/Cycle Framework

IBM* SAA* AD/Cycle* Language Environment/370 is a participant in the AD/Cycle framework, IBM's Systems Application Architecture* solution for developing and maintaining applications. The AD/Cycle framework consists of:

- *Tools* to support the full range of application development activities
- An *application development platform* of specifications and services to enable integration of those tools.

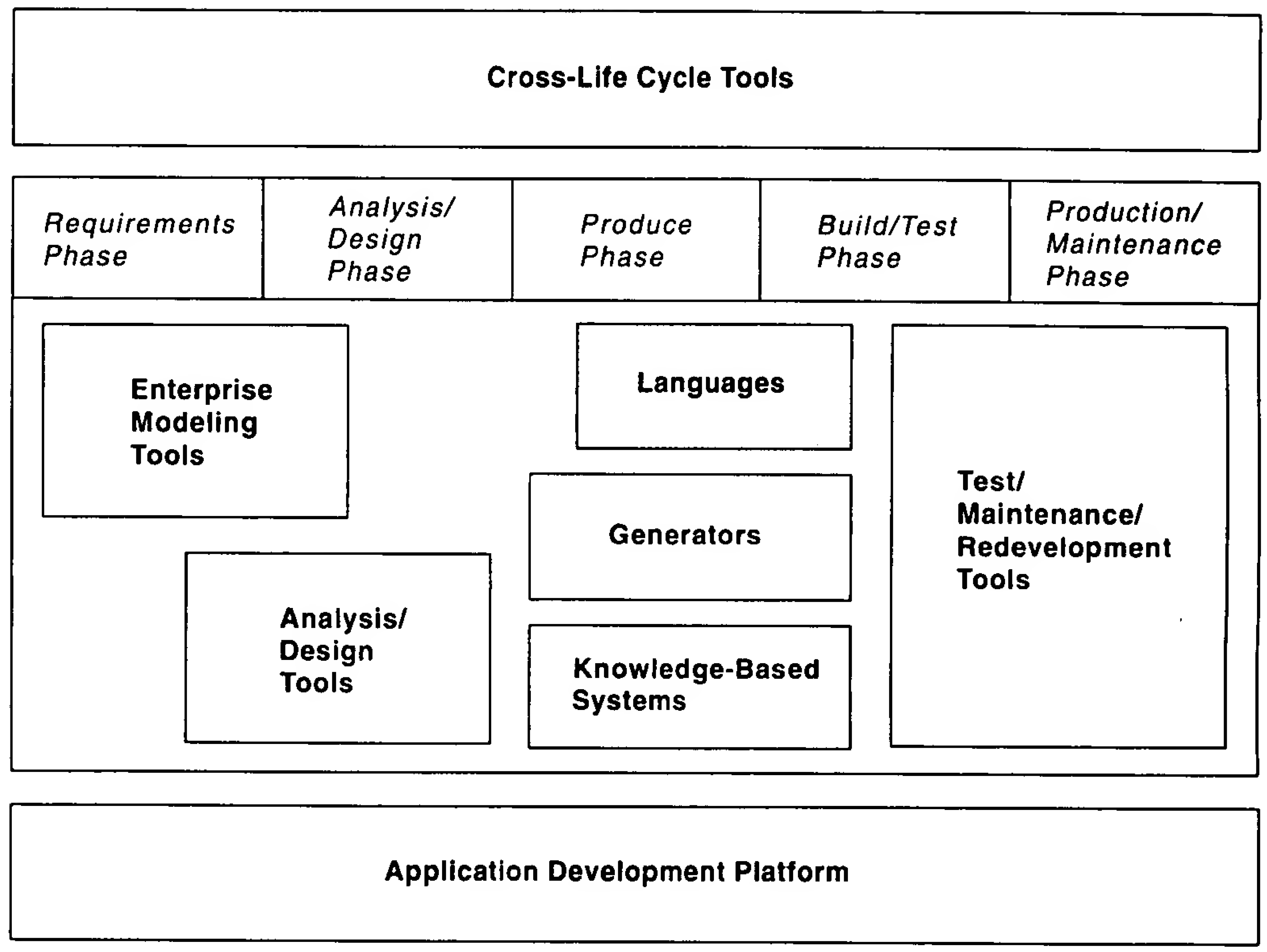


Figure 1. AD/Cycle Framework

AD/Cycle tools are grouped according to the type of application development activities they help perform. The boxes in Figure 1 show the AD/Cycle tool sets and their relationship to the traditional phases of application development. Also shown is the application development platform, which supports integration of the tools. LE/370 is part of the Languages tool set, as indicated by the shaded area.

For more information about the AD/Cycle framework, see the publication *Systems Application Architecture: AD/Cycle Concepts*, GC26-4531.

LE/370 fits into the produce and build/test phases of the traditional life cycle of application development and helps in the production/maintenance phase. LE/370 works with application generators or CASE tools that generate C and COBOL source code. When LE/370 is used in conjunction with other AD/Cycle standards, effectiveness of tools like Cross System* Product (CSP) and KnowledgeTool* is improved. You can reuse code, regardless of language, and mix your existing applications with LE/370-conforming ones. LE/370 reduces the restrictions put on the mixing of languages, allowing you to create applications largely from existing modules of code.

LE/370 is part of the Languages tool set, as shown in the following figure. Note the relationship of Language Environment, language compilers, and AD/Cycle CoOperative Development Environment (CODE).

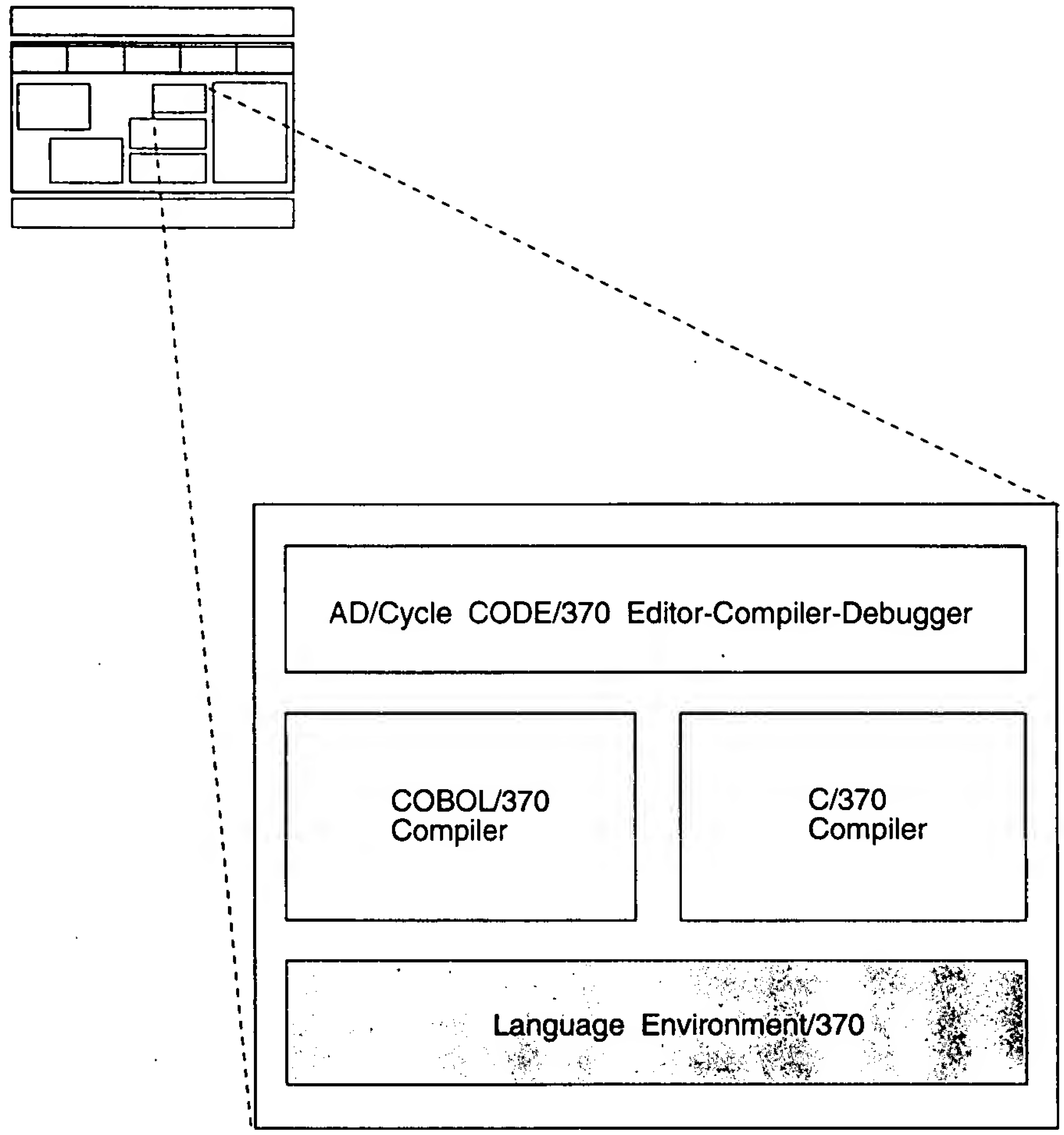


Figure 2. The Languages Tools Set in the AD/Cycle Framework

The AD/Cycle CODE Edit-Compile-Debug provides language-sensitive editing, and cooperative compiling, running, and advanced debugging function from the editor. This transparent interaction transfers some application development workload to the workstation. Editing and debugging interact with the host compiler to provide integrated editing, compiling, and debugging for application development and maintenance.

The language compilers provide the ability to program applications using the semantics particular to each language.

The Language Environment provides a common run-time environment for languages that conform to its architecture. To do this, it provides a common set of run-time options and services to assist with storage management, condition handling, and run-time message handling. It also allows interlanguage communication between languages conforming to its architecture, and describes a program model to define the operation of mixed language applications.

Programming Tasks and the Publications Library Structure

The AD/Cycle language products publications are separated based on the product separation described in the previous section. This structure is also designed to support the basic programming tasks outlined in the following figure.

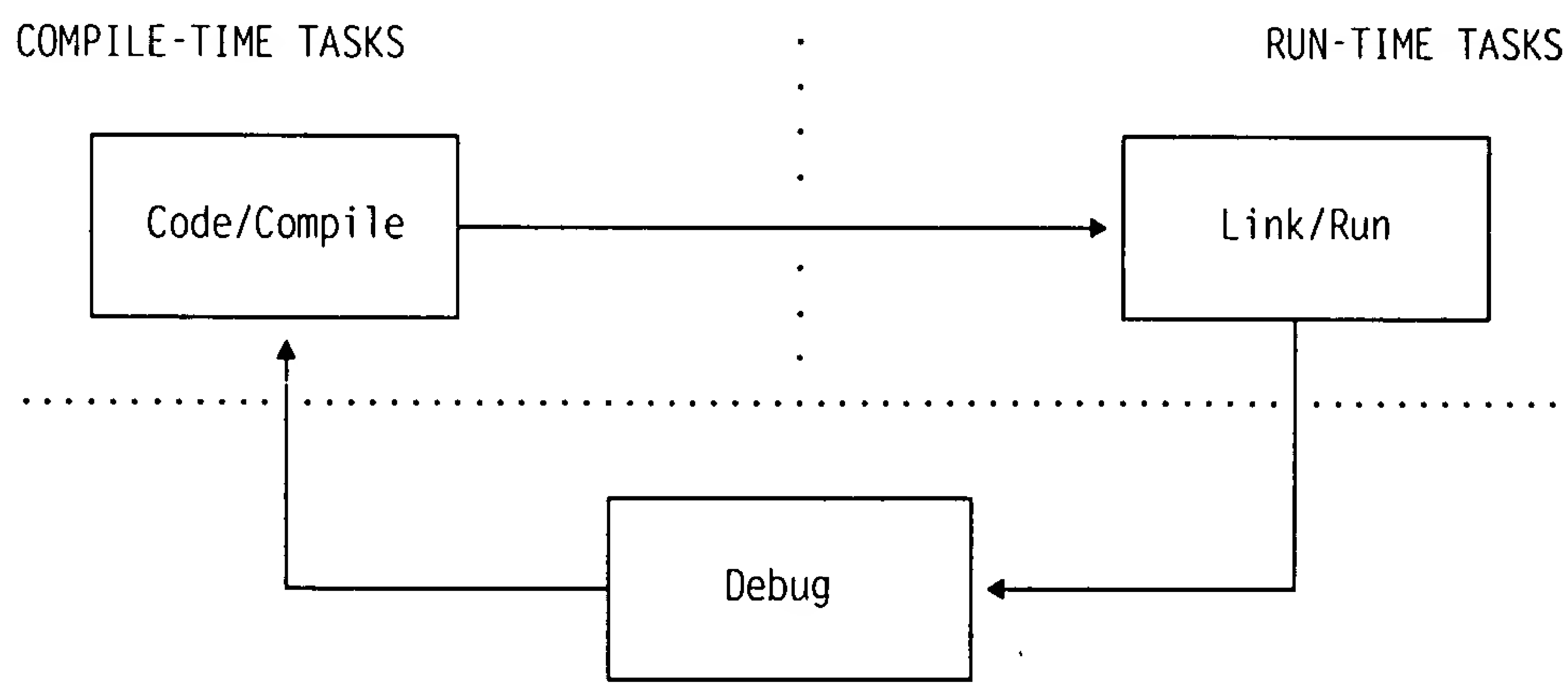


Figure 3. Division of Programming Tasks

- Publications supporting coding and compiling tasks are available with the language products you use.
- Publications supporting linking and running tasks are available with LE/370.
- Publications supporting debugging tasks are available with the AD/Cycle CODE/370 Debug Tool.

There are exceptions to this method of dividing tasks, but it is a good model to use in finding information. For example, an exception to this model would be that debugging tasks not using AD/Cycle CODE/370 can be found in both language compiler publications (for compile-time debugging information) and the LE/370 publications. Note also that interlanguage communication (ILC) is discussed mainly in the LE/370 publications.

Publications Provided with LE/370

Publications provided with LE/370 are the following:

Table 1. IBM SAA AD/Cycle Language Environment/370 Publications

Task	Publications	Order number
Evaluation and Planning	*Fact Sheet* *Concepts Guide* *Planning for Installation and Customization*	GC26-4785 GC26-4786 SC26-4817
Programming	*Programming Guide* *Debugging and Run-Time Messages Guide*	SC26-4818 SC26-4829
Diagnosis	*Diagnosis Guide*	LY37-3711
Warranty	*Licensed Program Specifications*	GC26-4774

LE/370 Publications Description

Fact Sheet
: Contains a brief overview and description of LE/370, C/370, and COBOL/370.

Concepts Guide
: Provides a detailed overview of program models and intended architecture for LE/370.

Planning for Installation and Customization
: Contains information needed to plan ahead for installing and customizing the LE/370 product.

Programming Guide
: Provides detailed information on:

  - Directions for linking and running programs that use LE/370 services
  - Information on storage management, run-time message handling, and condition handling models
  - Callable services and run-time options and how to use them
  - Instructions for writing programs that use interlanguage communication (ILC).

  This book also contains language-specific run-time information.

Debugging and Run-Time Messages Guide
: Provides detailed information on debugging techniques and services. Provides a listing of run-time messages and their explanations, as well as ABEND codes.

Diagnosis Guide
: Describes the procedures for creating a keyword string and reporting errors to IBM Service.

Licensed Program Specifications
: Contains a product description and warranty information.

For a complete list of publications you may need, see "Bibliography" on page 477.

How to Read the Syntax Diagrams

The following rules apply to the notation used in the syntax diagrams contained in this book:

- Read the syntax diagrams from left to right, top to bottom following the path of the line.
- Each syntax diagram begins with a double arrowhead (▸▸).
- An arrow (—▸) at the end of a line indicates that the option, service, or macro syntax continues on the next line. A continuation line begins with an arrow (▸—).
- IBM-supplied default choices are **boldface** and <u>underscored</u>.
- Keywords appear in non-italic capital letters and should be entered exactly as shown. However, some keywords may be abbreviated by truncation from the right as long as the result is unambiguous. In this case, the unambiguous truncation is shown in capital letters in the keyword, for example:

```
ANyheap
```

- Words in lower-case letters represent user-defined parameters or suboptions.
- Enter parentheses, arithmetic symbols, colons, semicolons, commas and greater-than signs where shown.
- Required parameters appear on the same horizontal line (the main path) as the option, service, or macro:

```
►►────OPTION───required_parameter──────────────────────────────►◄
```

- If you can choose from two or more parameters, the choices are stacked one above the other.

  If choosing one of the items is optional, the entire stack appears below the main line.

```
►►────OPTION──┬──────────────────────┬─────────────────────────►◄
              ├─optional_parameter_1─┤
              ├─optional_parameter_2─┤
              └─optional_parameter_3─┘
```

  If you *must* choose one of the items, one item of the stack appears on the main path:

```
►►────OPTION───┬─required_parameter_1─┬────────────────────────►◄
               ├─required_parameter_2─┤
               └─required_parameter_3─┘
```

- An arrow returning to the left above the a line indicates that an item can be repeated:

```
               ┌────────────────┐
               ↓                │
►►────OPTION──────repeatable_item┴──────────────────────────────►◄
```

OR

```
►►────OPTION──┬────────────────────┬────────────────────────────►◄
              │  ┌──────────────┐  │
              │  ↓              │  │
              └───repeatable_item┴─┘
```

- A comma or semicolon included in the repeat symbol indicates a separator that you must include between repeated parameters. These separators must be coded where shown.
- When entering commands, parameters and keywords must be separated by at least one blank if there is no intervening punctuation.
- A double arrow (→◄) at the end of a line indicates the end of the syntax diagram.

The following example demonstrates how to read the syntax notation. Numbers in the example correspond to explanations supplied below the example.

Format

```
       [1]
▶▶───ANyheap─┬──────────────────────────────────────────────────────────────────▶
             └─(──┬─────────────┬─,─┬────────────┬─,─┬────────────┬─,─▶
              [2] └─init_size───┘   └─incr_size──┘   ├─ANYwhere───┤
                      [3]                            └───BELOW────┘

▶─────────────────────┬─────────────────────────────────────────────────▶◀
▶───┬────────┬────)───┘
    ├─KEEP───┤ [4]
    └─FREE───┘ [5]
```

[1] Keyword with minimum unambiguous truncation shown in capital letters

[2] Opening parenthesis (must be specified if any parameters are specified)

[3] Optional parameter

[4] Optional keyword

[5] Optional keyword (IBM-supplied default)

Introduction to LE/370

Chapter 1. LE/370 Overview 2
What is LE/370? 2
Benefits of a Common Run-time Environment 2
Compatibility 3

Chapter 2. Program Model 4
Program Model Description 4
Terminology 5
  Process 5
  Enclaves 5
  Threads 7
Summary of Resource Ownership 8

© Copyright IBM Corp. 1991

Chapter 1. LE/370 Overview

This section provides an overview of LE/370 and the common run-time environment and introduces a number of frequently used LE/370 terms. Note that in this chapter and elsewhere in the book, the first usage of every such term is indicated in *italics*. You can find definitions for each italicized term in "LE/370 Glossary" on page 479.

What is LE/370?

LE/370 is a product that establishes a common run-time environment and common run-time services for language products, user programs, and other products.

The common run-time environment is comprised of data items and services performed by library routines available to a particular application running in the environment. The services that LE/370 provides to your application may include:

- Services that satisfy basic requirements common to most applications. These include support for the initialization and termination of applications, allocation of storage, support for ILC, and condition handling.
- Extended services often needed by applications. These functions are contained within a library of callable routines and include interfaces to operating system functions and a variety of other commonly used functions.
- Run-time options that help execution, performance, and diagnosis of your application.
- Access to operating system services.
- Access to language-specific library routines.

Benefits of a Common Run-time Environment

A common run-time environment offers:

- Common conventions across *LE/370-conforming* High Level Languages (HLLs).
- Common *callable services* across HLLs
- Common run-time options across HLLs
- A common program model
- A common run-time error message file
- User-controlled condition management
- A correspondence between conditions and messages delivered to you. For every condition in LE/370, there is a unique message.

Language Environment/370 Version 1 supports the following HLLs:

- IBM SAA AD/Cycle C/370
- IBM SAA AD/Cycle COBOL/370

Previous versions of these language products are supported for compatibility under LE/370.

LE/370 offers greatly improved consistency in the environment where your application runs by setting standards for a variety of commonly accessed functions. Run-time functions such as storage management, initialization, termination, program

© Copyright IBM Corp. 1991

management, condition handling, and math routines are handled by LE/370 in a common way across LE/370-conforming languages. Under LE/370 there is greater control over applications because you use these functions in a reliable and predictable manner in different language environments.

LE/370 provides a set of interfaces called callable services that allow you to use many of the LE/370 functions previously described and tailor them to the requirements of your application. You can access these services by including calls to LE/370 library routines in your application code. Calls to LE/370 routines generally have the same syntax and semantics across supported HLLs.

LE/370 also defines calling conventions that standardize the way programs call each other. Multiple-language applications written in LE/370-conforming HLLs gain improved ILC. Register conventions, parameter list formats, and data types conform to a common standard, making the behavior of ILC applications more predictable and consistent.

The behavior of LE/370 across all supported subsystems is also consistent. Under LE/370, applications running interactively under CICS or IMS generally behave in a manner consistent with those running in batch mode.

Compatibility

LE/370 provides support for COBOL and C/370 object and load module compatibility with existing language products. For most existing single language applications, you can:

- Run load modules in LE/370 that were compiled and linked under an earlier non LE/370-conforming version of the language.
- Run old object modules without recompiling, but with a relink.

If your existing application contains ILC, you must relink it in order to run it in the common run-time environment and take advantage of improved LE/370 ILC support.

Other exceptions will be noted where applicable in this book.

Detailed migration and compatibility information is described in the *Migration Guide* for each LE/370-conforming HLL listed in “Bibliography” on page 477.

Chapter 2. Program Model

The following sections provide an overview of the LE/370 program model. The model is key to understanding how the functions of LE/370 support your application.

The initial release of LE/370 implements a subset of this model. Features not supported in Language Environment/370 Version 1 Release 1 will be clearly indicated below and elsewhere in this document.

Program Model Description

LE/370 is best understood as part of a program model that regulates the behavior of applications composed of single and multiple HLLs such as COBOL/370 and C/370. LE/370 provides a common run-time environment and run-time services for these applications. The LE/370 program model supports the language semantics of applications which run in the common run-time environment and defines the way that routines or programs are put together to form an application.

Figure 4 illustrates the relationship between the various components that make up the LE/370 program model.

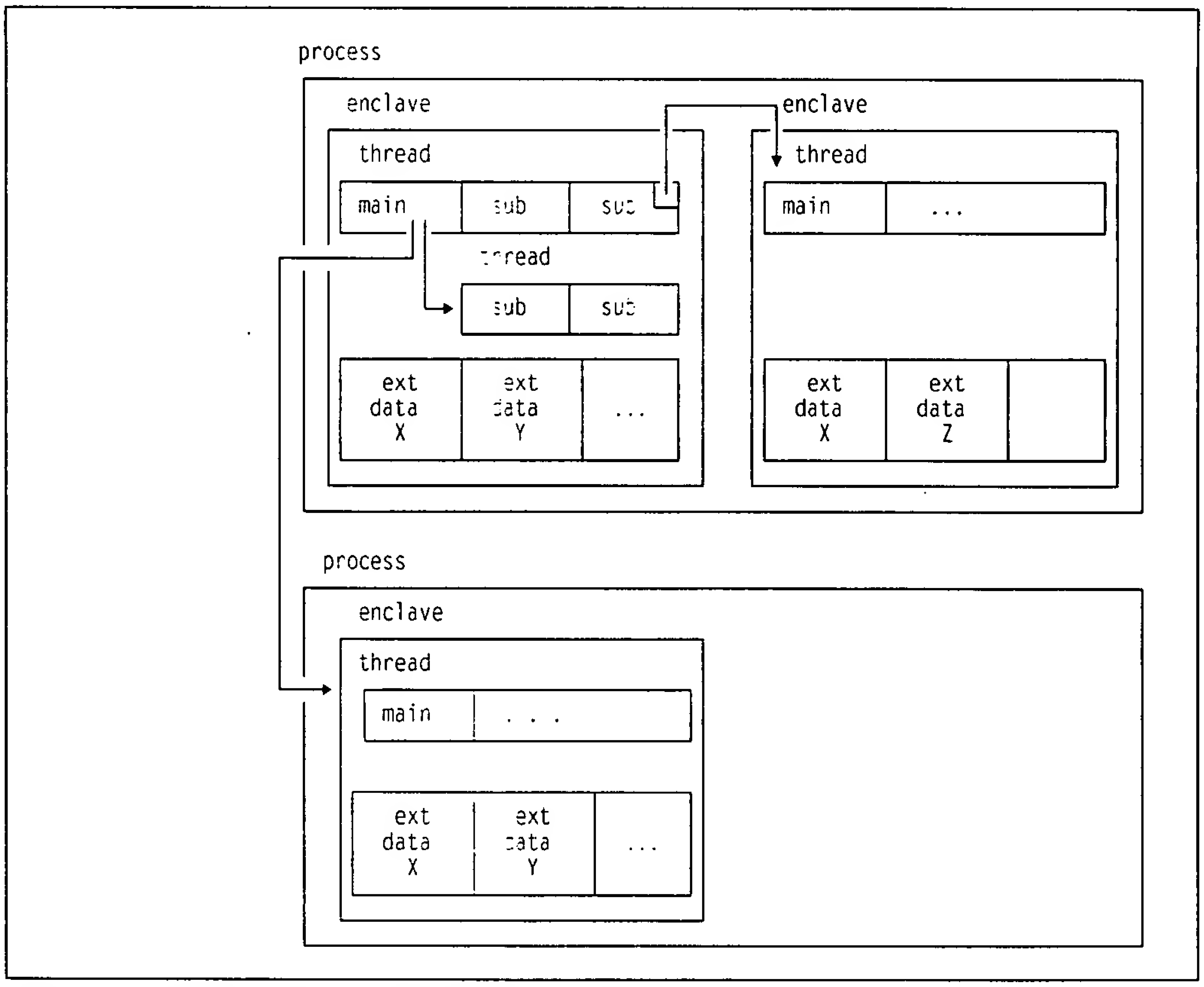


Figure 4. Overview of the LE/370 Program Model

© Copyright IBM Corp. 1991

Terminology

Three key program management constructs of the LE/370 program model shown in Figure 4 on page 4 are *processes*, *enclaves*, and *threads*.

Process

A *process* is a collection of resources, both application code and data, consisting of one or more related *enclaves*. As illustrated in Figure 4 on page 4, the process is the outermost run-time structure whose characteristics are described by the common run-time environment. The resources maintained at the process level do not affect the language semantics of an application executing at the *enclave* level.

The LE/370 library itself is an example of the type of resource that is maintained at the process level. The LE/370 library is loaded at process initialization, although it could be loaded for any of the individual enclaves within the process at enclave initialization. The process is used in the same way by all enclaves created within the process. It has no effect on the HLL semantics of applications executing within each of these enclaves.

Each process has an associated address space that is logically separate from those of other processes. Other than communicating with each other using certain LE/370 mechanisms, no resources are shared between processes. For example, processes do not share storage.

A process may create other processes. Note, however, that all processes are independent of and equal to one another. That is, they are not hierarchically related.

Note: Although the LE/370 program model supports applications consisting of one or more processes and one or more enclaves, Language Environment/370 Version 1 supports only a single process for each application executed in the common run-time environment. In addition, a single enclave is generally supported within the process (for exceptions to this rule, please see Chapter 29, "Nested Enclaves" on page 202). Therefore, bear in mind that any discussion of multiple processes and enclaves in this document is intended only to better define the boundaries of both.

Enclaves

A key feature of the program model is the *enclave*.

An enclave consists of one or more load modules, each containing one or more separately compiled, bound routines. In this book, the term *routine* is used as an exact equivalent of a COBOL/370 *compilation unit* or C/370 *function* or program, and means a named external routine, with or without named entry points, and with or without internal (contained) routines. The enclave is a logical run-time structure that supports the execution of a group of routines. A load module can exist in a variety of forms. It can be a mixture of HLL and/or assembler routines, combined with LE/370 routines.

An enclave in LE/370 is roughly analogous to:

- A *run-unit* in COBOL/370
- An instance of a `main()` function in C/370.

The enclave defines the scope of HLL semantics: for example, names, external data sharing, and control statements such as COBOL/370's STOP RUN, and C/370's `exit()`. The scope of each of these is discussed below:

- **The scope of the definition of the main and subroutines**

  The enclave boundary defines whether a routine is a *main* or *sub*. The first routine to execute in the enclave is known as the *main* routine in LE/370. All others are designated *sub*routines of the main routine.

  The first routine invoked in the enclave must be capable of being designated main according to the rules of the language of the routine. Conversely, all other routines invoked in the enclave must be capable of being sub according to the rules of the languages of the routines.

  Note that if a routine is capable of being invoked as either a main or sub, and recursive invocations are allowed according to the rules of the language, the routine may be invoked multiple times within the enclave. The first of these invocations could be as a main routine and the others as subroutines.

- **The scope and visibility of the following types of data**

  - *Automatic data*: Automatic data is data that is allocated with the same value upon entry and reentry into a routine. Values of the data at exit from the routine are not retained for the next entry into the routine. The scope of automatic data is a routine invocation within an enclave.
  - *External data*: External data persists over the lifetime of an enclave and maintains last-used values whenever a routine is reentered. The scope of external data is that of the enclosing enclave; all routines invoked within the enclave recognize the external data. Examples are COBOL/370 *external data* and C/370 external variables with static storage.
  - *Local data*: The scope of local data is that of the enclosing enclave; however, local data is recognized only by the routine that defines it. Examples are COBOL/370 *working storage* and any C/370 variable with block scope.

- **The scope of language statements**

  The enclave defines the scope of the language statements, for instance, those that terminate execution of the outermost routine within the enclave known to the language. COBOL/370's STOP RUN and C/370's `exit()` are examples. When one of these statements is executed, the main routine within the enclave terminates. Thus, the enclave defines the scope of the language statements.

  Prior to returning, resources obtained by the routines in the enclave are released and any open files (other than the LE/370 message file) are closed.

Additional Enclave Characteristics

Management of Resources: Most LE/370 resources, other than the message file, are managed by the enclave. For example, heap storage is shared among all routines within an enclave. Allocated heap storage remains allocated until explicitly freed, or until the enclave terminates.

None of the enclave-managed resources are shared between enclaves.

Multiple enclaves: Language Environment/370 Version 1 provides support for a single process within a single enclave and a single thread. Under some circumstances, however, it is possible for one enclave to pass control to others. For more information, see Chapter 29, "Nested Enclaves" on page 202.

Threads

Within each enclave is a *thread* of execution. A thread is a logical execution path represented by the machine state; conditions raised during execution are isolated to that execution path.

The thread is the basic line of execution within the LE/370 program model. It is dispatched with its own instruction counter and registers by the system. Threads can execute concurrently with other threads.

Threads share all of the resources of an enclave and therefore do not need to selectively create or load new copies of resources, code, or data. Although a thread does not own its storage, it can address all storage within the enclave. A thread, however, does have its own separate stack for automatic data and a distinct condition manager. All threads are equal to and independent of one another and are not related hierarchically.

The LE/370 program model supports multiple threads within an enclave.[1] Multiple threads provide support for parallel processing. They allow an application to split into multiple concurrent paths of execution, potentially shortening the time required to complete a large task on a machine capable of parallel processing.

For example, a very large array could be processed by multiple threads within a single enclave. If the array had 100,000 elements, the first thread could perform some processing on elements 1 to 10,000; the second could perform concurrently the same processing on elements 10,001 to 20,000; and so on. Note that during the processing, each thread would maintain its own execution state, storage, and condition manager. A condition raised in one of the threads would be handled by that thread and would not affect the execution of any other thread performing the parallel processing.

[1] In its first release, LE/370 supports only a single thread within an enclave. Therefore, parallel processing is not supported in Release 1.

Summary of Resource Ownership

Figure 5 provides an overview of resource ownership in Language Environment/370 Version 1 using the terminology defined above. Whenever you run a routine under LE/370, a process, an enclave, and an initial thread are all implicitly created. Figure 5 shows logical ownership of resources by each component that comprises the LE/370 program model.

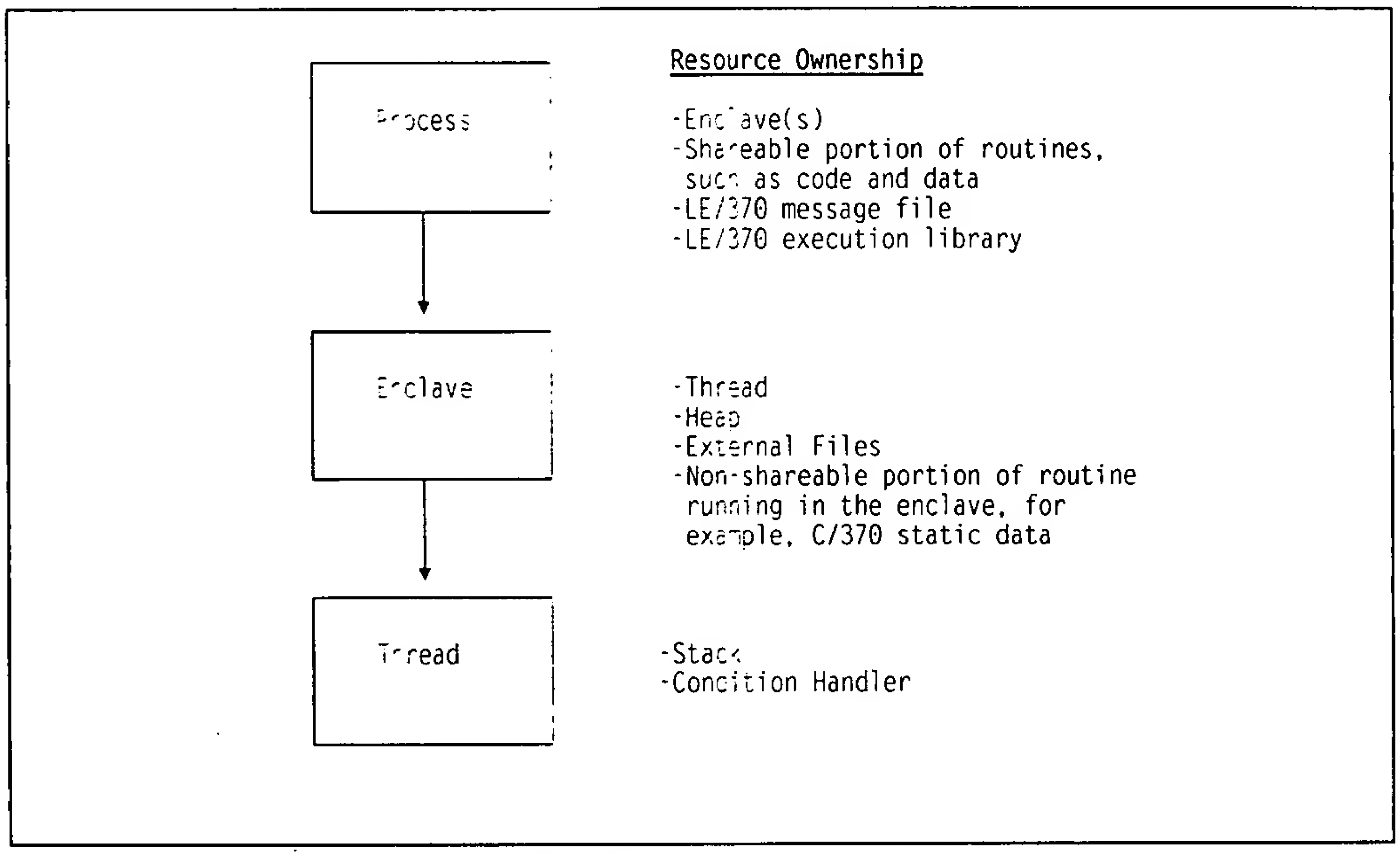


Figure 5. Overview of Resource Ownership

Considerations for Writing an LE/370 Application

Coding an application is generally the same under an LE/370-conforming language as in earlier versions of the language. However, in order to take advantage of some of the features that a common execution environment offers, consider a number of things when writing an LE/370 application (particularly one that relies on ILC). For example, LE/370 imposes a common return/reason code scheme that can differ from a version of the language prior to the LE/370-conforming version. LE/370 offers new global and local user exits that can include functions in addition to those you are accustomed to.

When running in the common run-time environment, you must also consider the target operating system. Currently, under CMS, MVS, TSO, CICS and IMS, the way that parameters are passed differs. To ensure consistency, LE/370 attempts to standardize the parameter list format across operating systems as much as possible. It is therefore important for you to know what LE/370 does to the format to ensure this consistency.

The following chapters describe these considerations:

Chapter 3. Parameter List Formats 10
Argument Lists and Parameter Lists 10
Methods for Passing Arguments to and from Routines 10
Format of Arguments Passed by Operating Systems and Subsystems 12
C/370 Parameter Passing Styles 14
  PLIST and EXECOPS Interactions 17

Chapter 4. Application Return Codes 20
Application Terminating Events 20
Enclave Return Code Processing 21

Chapter 5. Run-time User Exits 23
  User Exits for Initialization 24
  User Exit for Termination 24

© Copyright IBM Corp. 1991

Chapter 3. Parameter List Formats

One of the major things to consider when writing an LE/370-conforming application is how parameters are passed to the application upon invocation. The type of parameter list created for the application varies according to the operating system or subsystem being used.

This chapter describes passing parameters to external routines under LE/370. The methods described do not apply to internal routines or to compiled code invoking its own library routines. Each LE/370-conforming HLL may have its own method for transferring control and passing arguments between internal routines.

Argument Lists and Parameter Lists

The terminology used to describe passing parameters to and from routines currently differs among LE/370-conforming HLLs. Figure 6 summarizes the terminology used with LE/370. In Figure 6, a calling routine passes an *argument* list to a called routine. That same list is termed a *parameter* list when it is received by the called routine. Under LE/370, the formats of the argument and parameter lists are identical. The only difference between the two terms is whether it is being used from the point of view of the calling or the called routine.

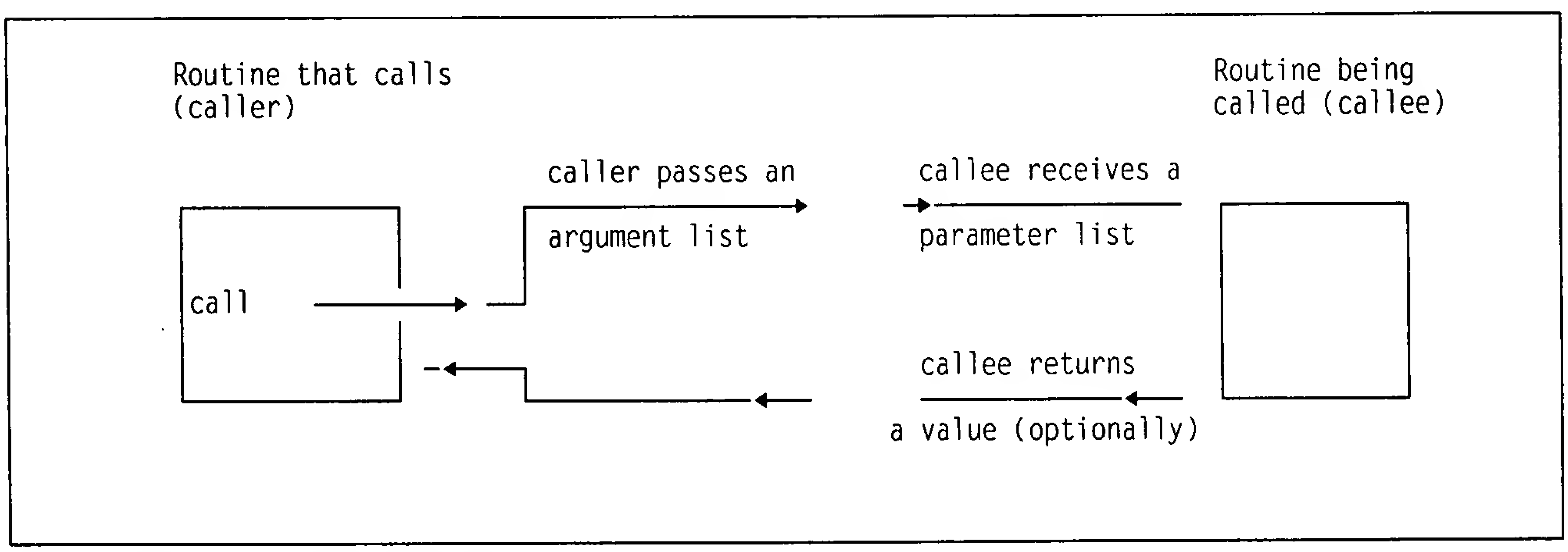


Figure 6. Call Terminology Refresher

Methods for Passing Arguments to and from Routines

Table 2. Semantic Terms and Methods for Passing Arguments in LE/370

semantic term method	**by value**	**by reference**
directly	A copy of the argument is made	Not allowed under LE/370
indirectly	A pointer points to a copy of the argument	A pointer points to the argument

© Copyright IBM Corp. 1991

Table 2 illustrates commonly used semantic terms and methods for passing arguments, and indicates what is actually passed to routines. LE/370-conforming HLLs use two basic semantic terms for passing arguments:

by value Only the value of the object is passed. Any changes made by the calling routine to the argument value will not be reflected in the calling routine.

by reference Changes made by the calling routine to the argument value will be reflected in the calling routine.

Likewise, under LE/370, there are two methods for passing arguments:

directly The value of the argument is placed directly in the argument list body.

indirectly The body of the argument list contains a pointer to the argument value.

Figure 7 illustrates these argument passing styles. In the figure, Register 1 (R1) points to a copy of an argument, to another pointer to a copy of the argument or to the real argument.

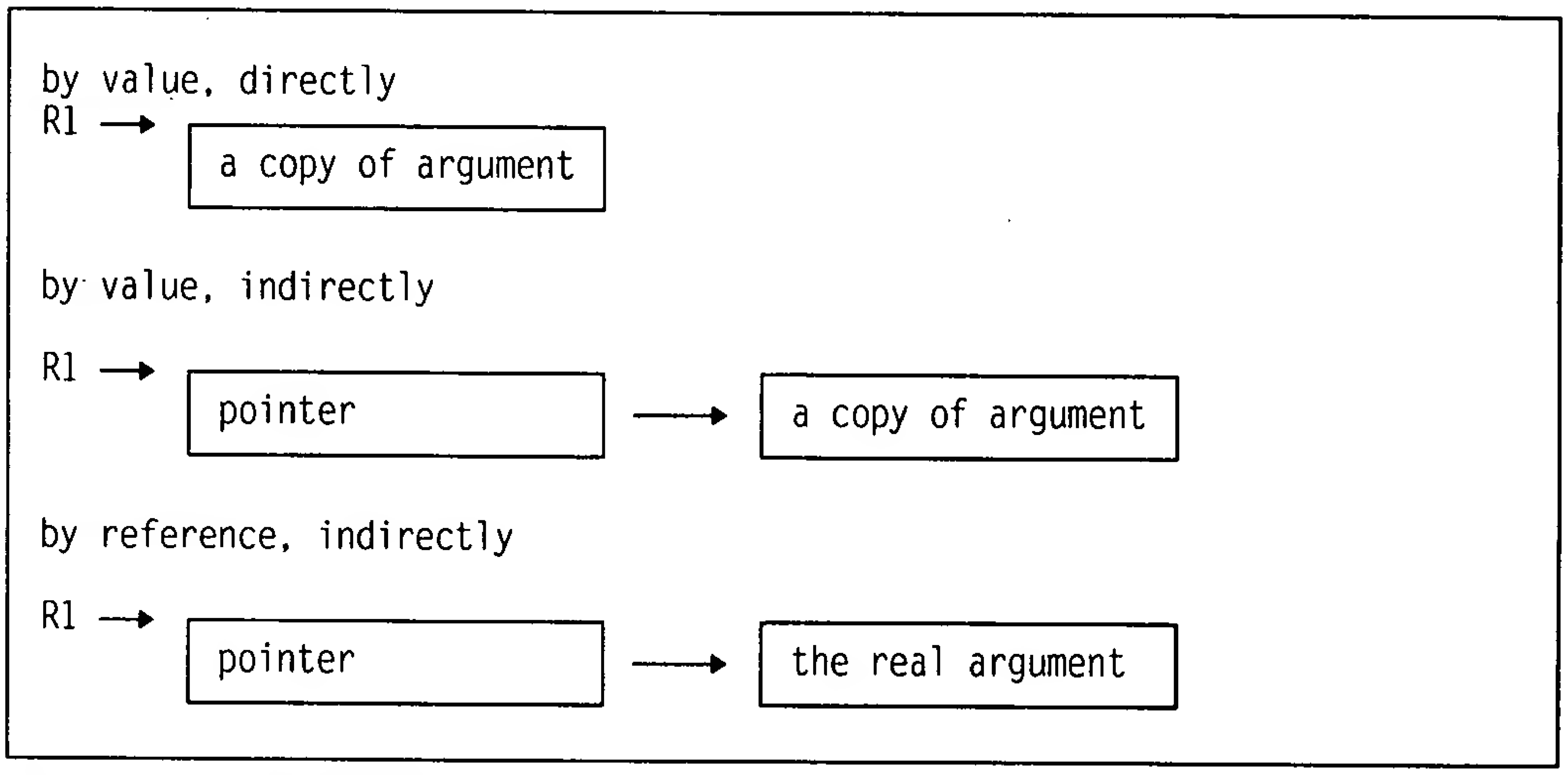


Figure 7. Argument Passing Styles in LE/370

You cannot pass an argument by reference, directly.

Note that the structure of passing arguments by value indirectly and by reference indirectly *appears* to be the same. The difference between the two methods is that the pointer in the parameter address list points in the first case to a copy of a parameter, and in the second case to the parameter itself.

HLL semantics usually determine when data is passed by value and when it is passed by reference. LE/370 supports the following argument passing styles for the following HLLs:

- C/370 passes arguments by value, directly
- COBOL/370 passes arguments
  - by reference, indirectly
  - and by value, indirectly (also known as *by content*).

Format of Arguments Passed by Operating Systems and Subsystems

HLL and assembler applications that accept operating system or subsystem parameters must know which parameter list format to expect when receiving parameter lists from these systems. For example, if you specify run-time options on the command line, your application needs to know the format of the parameter list containing the run-time options that the operating system will pass. The formats for System/370* (S/370) operating systems and subsystems are described below.

Under S/370, an argument list is located by a argument list pointer that is held in general purpose register 1 (R1) (see Figure 8). As indicated in the figure, the argument list body contains a list of addresses of the actual arguments.

```
argument          argument list body
list ------>    +-----------------------+
pointer         |  A ( Argument  1 )    |
(R1)            +-----------------------+
                |  A ( Argument  2 )    |
                +-----------------------+
                |                       |
```

Figure 8. S/370 Argument List Format

Depending on the operating system and subsystem, and the type of argument list being used, LE/370 takes one of two possible actions regarding argument format:

1. If the arguments are passed in the MVS argument list or CMS extended argument list formats, LE/370 repackages the arguments as follows:

   MVS Argument List Format

   The MVS argument list format is shown in Figure 9. Register 1 points to a fullword address with the high order bit on (indicated by the number 1 in the figure). This address holds the location of a halfword length prefix followed by a character string. This form of the argument is constructed by LE/370 under MVS and presented to the application.

   C Considerations: When the EXECOPS run-time option is specified under MVS, LE/370 alters the MVS parameter list format. LE/370 removes any run-time options that are present.

```
R1 ------> |1 A(var_ln_string)| ---------> |L L|    arguments     |
                                               |------LL----------|
Note: LL field is zero if there are no arguments
```

Figure 9. MVS Argument List Format (LE/370 potentially changes the host format)

   CMS Extended Argument List Format

   The CMS extended argument list format is shown in Figure 10 on page 13. In the figure, Register 0 points to a vector of fullword addresses, the second and third indicating the starting and ending addresses of the character parameter. This format is repackaged by LE/370 into the same format as shown in Figure 9.

   Note: The CMS tokenized argument list format is not supported under LE/370.

```
R0 ───────▶ ┌──────────────────┐
            │ A(command)       │
            ├──────────────────┤
            │ A(arg_start)     │
            ├──────────────────┤
            │ A(arg_end)       │
            ├──────────────────┤
            │ A(user_word)     │
            └──────────────────┘
Note: R0 is used rather than R1
```

Figure 10. CMS Extended Argument List (LE/370 Changes the Host Format)

2. If other argument list formats are being used, LE/370 passes the arguments as is to the application:

   OS Argument List Format

   The OS argument list is located by an argument list pointer that is held in general purpose register 1 (R1) (see Figure 11). As indicated in the figure, the argument list body contains a list of addresses of the actual arguments.

```
argument          argument list body
list  ───────▶  ┌────────────────────────┐
pointer         │ A ( Argument  1 )      │
(R1)            ├────────────────────────┤
                │ A ( Argument  2 )      │
                ├────────────────────────┤
                │                        │
```

Figure 11. S/370 Argument List Format

   IMS Argument List Format

   The IMS extended argument list format is shown in Figure 12. In the figure, Register 1 points to a vector of fullword addresses. Each address in turn points to an IMS Program Communication Block (PCB).

```
R1  ───────▶ ┌─────────────┐
             │ A(PCB_1)    │ ───────▶  PCB_1
             ├─────────────┤
             │ A(PCB_2)    │ ───────▶  PCB_2
             └─────────────┘
                  ...                  ...
             ┌─────────────┐
             │1 A(PCB_n)   │ ───────▶  PCB_n
             └─────────────┘
```

Figure 12. IMS Argument List Format - LE/370 Interface (Passed As Is)

CICS Argument List Format

The CICS argument list format is shown in Figure 13. In the figure, Register 1 points to the addresses of an *EIB* (EXEC Interface Block) and a *COMMAREA*.

```
R1 ----> | A(EIB_PTR)     | ------> | A(EIB)   | ------> EIB
         | 1 A(COMAR_PTR) | --+
                              +---> | A(COMAR) | ------> COMAR
```

Figure 13. CICS Argument List Format (Passed As Is)

TSO Argument List Format

The TSO argument list format is shown in Figure 14. In the figure, Register 1 points to a *CPPL* (Command Processor Parameter List).

```
R1 ----> | A(Command buffer)                 |
         | A(User profile table)             |
         | A(Protected step control block)   |
         | A(Environment control block)      |

Command buffer layout
  | Length | Offset | Text         //          |
  0        2        4
```

Figure 14. TSO Argument List Format

For more information about the IMS, CICS, or TSO parameter formats illustrated in the figures, refer to the applicable programming guide for each subsystem listed in "Bibliography" on page 477.

C/370 Parameter Passing Styles

This section applies to C/370 users only.

C/370 generally supports a single character string as a parameter to a main routine. It parses the string into tokens that are accessed by the `argc, argv` parameters of the main function.

In addition, there are alternate styles of passing a set of parameters to the main routine according to some other arrangement; for example, a single value, a pointer to a value, or a pointer to a list of values. In these cases, the set of parameters is not parsed. It is assumed that the invoker of the application (for example, the operating system) has stored the address of the set of parameters in Register 1 upon entry into the main routine. Depending on how the parameters passed, R1 points to the following upon entry, as illustrated in Figure 15 on page 15:

```
Style 1 - Register 1 contains parameter value

     Register 1 = parameter value


Style 2 - Register 1 contains pointer to parameter
          value

     Register 1 = pointer ——→ parameter value


Style 3 - Register 1 contains pointer to array
          of pointers to parameter values

     Register 1 = pointer ——→ (pointer0——→value0)
                              (pointer1——→value1)
                              (pointer2——→value2)
                                     .
                                     .
                                     .
                              (pointern——→valuen)
```

Figure 15. Some Alternate C/370 Parameter Passing Styles

Note that the first arrangement in Figure 15 can be used only for parameters that are integers.

A main routine elects to use one of the styles shown in Figure 15 by specifying the PLIST(OS) run-time option (see "PLIST and EXECOPS Interactions" on page 17). The main routine must know which particular parameter list style to expect. When PLIST(OS) is specified, C/370 makes the parameter list available through a pair of macros. Code these in your main routine to determine which parameter list style your routine will receive.

__R1 of type "void *"
: __R1 contains the value contained in Register 1 upon entry into the main routine. It provides access to the parameters when they are passed according to the first two styles shown in Figure 15.

__osplist of type "void **"
: __osplist acts as an array of pointers to parameters. It is derived from __R1 and provides access to the parameters when they are passed according to the third style shown in Figure 15.

Note: The third style is also currently supported with certain macros and functions (for example, __pcblist and __csplist) for certain invokers (for example, IMS and CSP). __osplist is a generalization of __pcblist and __csplist. It can be used in place of the more specialized __pcblist and __csplist macros and in cases where these macros don't apply.

Figure 16 on page 16 illustrates how these macros would be used to access items in the three alternate parameter arrangements.

```
Style 1

     Register 1 = __R1


Style 2

     Register 1 = __R1 ──→ *__R1


Style 3

     Register 1 = _R1 ──→(__osplist[0]──→*__osplist[0])
                         (__osplist[1]──→*__osplist[1])
                         (__osplist[2]──→*__osplist[2])
                                          .
                                          .
                                          .
                         (__osplist[n]──→*__osplist[n])
```

Figure 16. Accessing Parameters Using Macros

Note that suitable casting and dereferencing are required when using these macros. Figure 17 contains examples of the casting and dereferencing that must be used with these macros to access parameter values and/or pointers according to which parameter passing style is in use.

```
Style 1

     parm     = (int) __R1;   (restricted to integral types)


Style 2

     parm_ptr = (float *) __R1;
     parm     = * ((float *) __R1);


Style 3

     parm0_ptr = (float *) __osplist[0];
     parm0     = * ((float *) __osplist[0];
```

Figure 17. Examples of Casting and Dereferencing

PLIST and EXECOPS Interactions

C/370 #pragma runopts allows you to specify to the C/370 compiler a list of run-time options that are to be used at run-time. Two of the options on #pragma runopts affect the format of the argument list passed to the application upon initialization.

The EXECOPS option allows you to specify run-time options on the command line or in JCL at application invocation. ~~These options override the defaults established by CEEUOPT and~~ CEEDOPT, including those designated as nonoverrideable ~~(see Chapter 30, "Specifying Run-time Options" on page 206 for more information)~~. NOEXECOPS indicates that run-time options cannot be specified on the command line or in JCL at application invocation.

The PLIST option indicates how the invoked routine should anticipate the argument list. For example, if you specify PLIST(OS), the argument list that your application receives upon invocation is assumed to be in an OS format.

You can specify PLIST with the following values under LE/370:

HOST The argument list is assumed to be a character string. The string is located differently under various systems as follows:

- Under CMS, if invoked by OSRUN, use the string presented in an MVS-like format located by the pointer held in R1.
- Under CMS, if not invoked by OSRUN, use the CMS extended parameter list.
- Under TSO, if a CPPL is detected, obtain the string from the command buffer.
- Under TSO, if a CPPL is not detected, assume a halfword prefixed string in the MVS-format.
- Under MVS, use the halfword prefixed string.

OS The inbound parameter list is in an MVS linkage format in which R1 points to a parameter address list. No execution time options are available and R1 is not interrogated.

The PLIST(HOST) setting permits portability of source code across MVS and CMS. PLIST(HOST) allows the object to execute under CMS (using either the MVS-format argument list for OSRUN or the extended argument list), under MVS (assuming a halfword prefixed string), and under TSO (possibly using the CPPL or the MVS-format parameter list). Specify PLIST(HOST) in order to default to the argument list format for the operating system that your application is executing under.

Although LE/370 supports the MVS, CMS, IMS, and TSO suboptions of PLIST for compatibility, use of PLIST(HOST) is recommended. There are some exceptions to this guideline:

- **Pre-initialization** — In the pre LE/370-conforming C/370 interface to pre-initialization, it was necessary to specify PLIST(MVS) in order to flag pre-initialized routines. PLIST(MVS) is still supported for compatibility.
- **CICS** — If you are running an old CICS application compiled under the pre LE/370-conforming version of C/370, the default PLIST(HOST) is assumed regardless of the actual setting of PLIST. If you are running a CICS application compiled under IBM SAA AD/Cycle C/370, specify PLIST(OS).
- **TSO** — TSO command processors that require access to the full CPPL must specify PLIST(OS).

The interaction of the `EXECOPS|NOEXECOPS` and `PLIST` options can alter the format of the argument list passed to your application, depending on the combination of options specified. These interactions under CMS and MVS are summarized in Table 3, and Table 4 on page 19.

Table 3. Interactions of PLIST and EXECOPS under CMS

PLIST setting	NOEXECOPS	EXECOPS
PLIST(CMS)	Use the CMS extended parameter list and re-package in a halfword prefixed string without looking for run-time options.	Use the CMS extended parameter list, remove any run-time options, and re-package in a halfword prefixed string.
PLIST(HOST)	If invoked by OSRUN, pass R1 through without change and without looking for run-time options. If not invoked by OSRUN, assume R0 points to a CMS extended parameter list. Re-package as a halfword prefixed string without looking for run-time options.	If invoked by OSRUN, assume a halfword prefixed string is passed. Check for run-time options and remove them. Pass an adjusted string (without run-time options) to the application. If not invoked by OSRUN, assume R0 points to a CMS extended parameter list. Remove any run-time options and re-package as a halfword prefixed string.
PLIST(MVS)	Use R1 assuming a halfword prefixed string, passing the entire string to the main routine without change (that is, simply pass R1 unaltered).	Use R1 assuming a halfword prefixed string. Remove any run-time options from the string and pass the (potentially) altered string to the main routine.
PLIST(IMS)	Pass R1 through without change.	Pass R1 through without change.
PLIST(CICS)	Pass R1 through without change.	Pass R1 through without change.
PLIST(TSO)	Add a level of indirection to the parameter list.	Add a level of indirection to the parameter list.
PLIST(OS)	Pass R1 through without change.	Pass R1 through without change.

Note: When LE/370 is directed to use the CMS extended parameter list, and LE/370 determines that R0 is **not** pointing to a CMS extended parameter list, LE/370 will issue user ABEND 4093, reason code 16.

Table 4. Interactions of PLIST and EXECOPS under MVS

PLIST setting	NOEXECOPS	EXECOPS
PLIST(CMS)	This is an error. ABEND 4093, reason 16.	This is an error. ABEND 4093, reason 16.
PLIST(HOST)	If executing under TSO and LE/370 determines that a CPPL has been passed, the command buffer is used to extract the inbound character string. Construct a halfword prefixed string to pass to the application without looking for run-time options. If not executing under TSO, or if executing under TSO and a CPPL is not discovered, pass R1 into the application unaltered without looking for run-time options.	If executing under TSO and LE/370 determines that a CPPL has been passed, the command buffer is used to extract the inbound character string. Construct a halfword prefixed string to pass to the application after run-time options have been removed. If not executing under TSO, or if executing under TSO and a CPPL is not discovered, remove run-time options and pass a (possibly) re-adjusted R1 into the application.
PLIST(MVS)	Use R1 assuming a halfword prefixed string, passing the entire string to the main routine without change (that is, pass R1 unaltered).	Use R1 assuming a halfword prefixed string. Remove any run-time options from the string and pass the (potentially) altered string to the main routine.
PLIST(IMS)	Pass R1 through without change.	Pass R1 through without change.
PLIST(CICS)	Pass R1 through without change.	Pass R1 through without change.
PLIST(TSO)	Add a level of indirection to the parameter list.	Add a level of indirection to the parameter list.
PLIST(OS)	Pass R1 through without change.	Pass R1 through without change.

Chapter 4. Application Return Codes

In the past, many HLLs (and in particular, C/370) handled conditions that occur in the run-time environment using a *return code*-based model. LE/370, on the other hand, manages conditions according to a *condition*-based model that is described in Chapter 19, "Condition Management" on page 83. While you are encouraged to make use of all of the condition handling mechanisms that are offered by this model, LE/370 also provides consistent support for existing applications that rely on a return code-based condition handling scheme.

Under LE/370, a return and reason code are returned to the invoker of an enclave when the enclave terminates. Because Language Environment/370 Version 1 generally supports a single enclave running within a single process, termination of the enclave generally means that your application has terminated. For exceptions to this rule, see Chapter 29, "Nested Enclaves" on page 202. If you invoked the enclave from the command line, JCL, or catalogued procedures, the return and reason code are returned to the operating system.

Application Terminating Events

Several events can terminate an application and generate a return code. These include:

Return from a main routine
When termination is due to a normal return from a main routine, the return code is set as HLL semantics dictate:

- In COBOL, by a GOBACK from the main routine
- In C, by a `return()` from the main routine.

Language STOP construct
When termination is due to one of the following language constructs, the enclave user return code is set as HLL semantics dictate:

- In COBOL, by a STOP RUN statement.

  When you issue a STOP RUN in COBOL, the user return code is taken from COBOL's RETURN-CODE special register.

  See the *IBM SAA AD/Cycle COBOL/370 Programming Guide* for more information about this language construct.

- In C/370, as indicated in Table 5.

Table 5. C/370 Language Constructs That Generate a Return Code

Termination by	User Return Code
exit()	a fullword binary value (–2**31 to +2**31 – 1)
sigfpe()	3000
sigill()	3000
sigsegv()	3000
sigabrt()	2000
sigterm()	0

© Copyright IBM Corp. 1991

See the *IBM SAA AD/Cycle C/370 Programming Guide* for more information about these language constructs.

unhandled condition (severity 2 or above)
When termination is the result of an unhandled condition of severity 2 or above on the last thread in the enclave, LE/370 implicitly creates a user return code of 0 and an LE/370 return code modifier as described in the following section.

Enclave Return Code Processing

When an application terminates, the return code that is presented to the operating system is calculated as indicated in Figure 18.

```
Enclave return code = user return code + LE/370 return code modifier
```

Figure 18. Calculation for Enclave Return Code

where

user return code
is set by an HLL construct (such as the RETURN-CODE special register in COBOL).

LE/370 return code modifier
is set by LE/370 routines that manage the environment. The LE/370 return code modifier contains an LE/370 severity code which indicates how the enclave terminated. The LE/370 severity code is initially set to 0, indicating normal enclave termination. It can also be set to 2, 3, or 4 based on the severity of an unhandled condition that caused the enclave to terminate, as indicated in Table 6 on page 22.

For more information about LE/370 conditions and severity codes, see Chapter 19, "Condition Management" on page 83.

The LE/370 return code modifier is calculated by multiplying the severity code by a value based on whether the application terminated on an MVS or non-MVS system. Under MVS, the severity code is multiplied by 1000. In non-MVS environments, it is modified by 10**6.

When the enclave terminates, the LE/370 return code modifier is placed in R0. The user return code and the LE/370 return code modifier are summed to comprise the enclave return code and placed into R15.

Summary of LE/370 Reason Codes

Table 6 on page 22 contains a summary of Reason Codes produced when a routine is called from the command line or when the initial routine within an enclave terminates.

Table 6. Summary of LE/370 Reason Codes

Condition Severity	Meaning	LE/370 Return Code Modifier — MVS(R0)	LE/370 Return Code Modifier — non-MVS(R0)
0	Normal application termination	0	0
Unhandled severity 2 condition	Error — abnormal termination	2000	2,000,000
Unhandled severity 3 condition	Severe error — abnormal termination	3000	3,000,000
Unhandled severity 4 condition	Critical error — abnormal termination	4000	4,000,000

CMS Considerations: **Warning:** The CMS Ready(XXXXX) prompt displays only the last five digits of the enclave return code. Under CMS, some enclave return codes containing more than 5 digits (for example, 2,000,000 or 3,000,000) will *not* be displayed. In this case, the CMS Ready prompt will indicate the following:

```
Ready(00000);
```

C Considerations: The LE/370 enclave return codes are incompatible with the return codes returned under the pre LE/370-conforming version of C. For example, `abort()` previously returned 1,000, but now returns 2,000,000 under CMS.

Chapter 5. Run-time User Exits

LE/370 provides two user exits, one written in Assembler and the other in an LE/370-conforming HLL, that you can modify when you run an application. The user exits offer you a chance to perform certain functions at a point where you would not otherwise have a chance to do so. For example, you can specify a list of run-time options in an assembler initialization user exit that establish characteristics of the environment. This is done prior to the actual execution of any of your application code. Another example is using an assembler user exit for termination to request a dump after your application has terminated with an ABEND.

Figure 19 shows the location of user exits at initialization and termination processing.

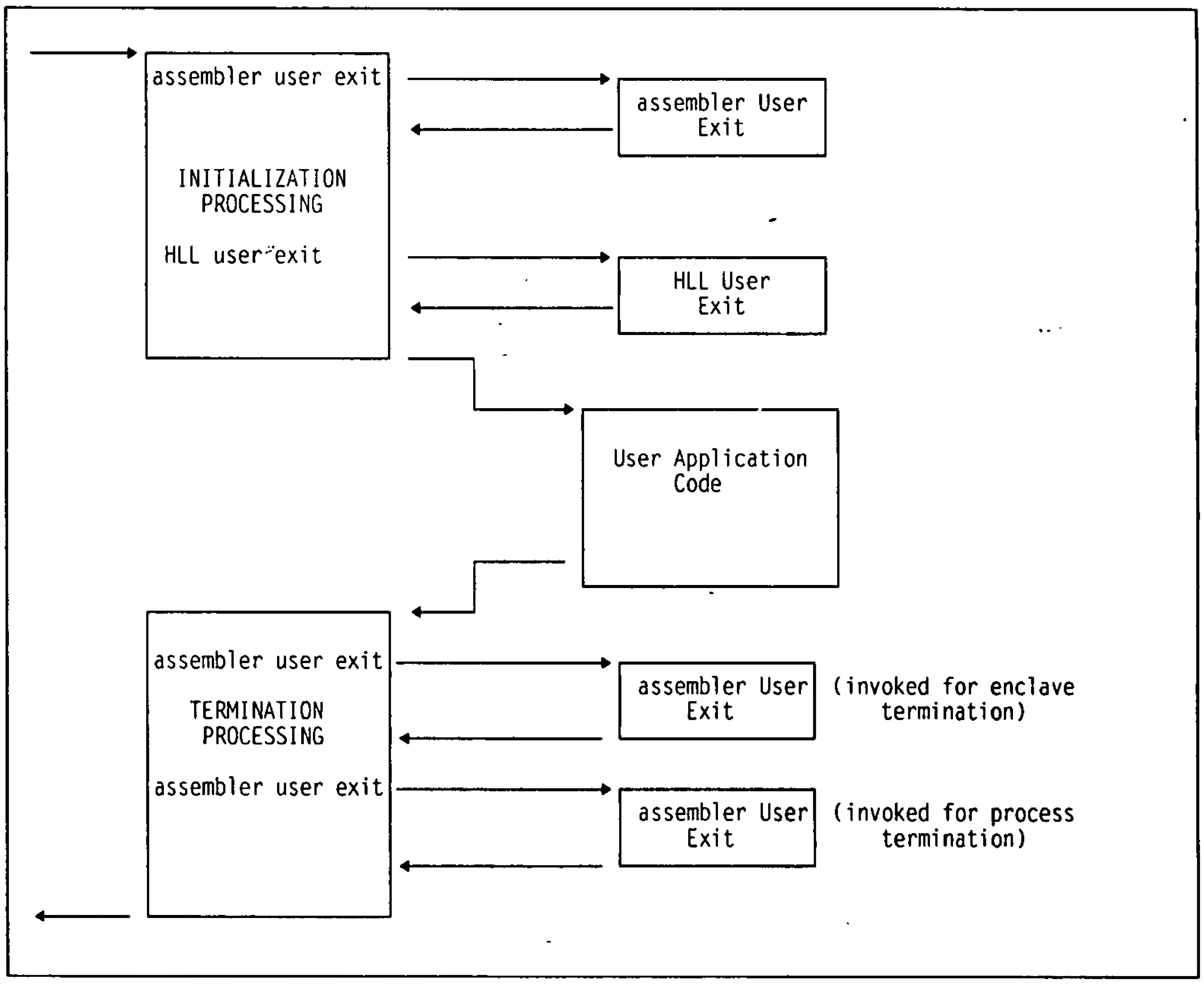


Figure 19. Location of User Exits

In Figure 19, run-time user exits are invoked in the following sequence:

1. Assembler user exit invoked for enclave initialization
2. Environment is established
3. HLL user exit invoked
4. Main routine invoked
5. Main routine returns control to caller
6. Environment is terminated
7. Assembler user exit invoked for termination of the enclave
8. Assembler user exit invoked for termination of the process.

© Copyright IBM Corp. 1991

User Exits for Initialization

IBM-supplied default versions of the assembler and HLL initialization user exits are provided for each operating system. The assembler user exit (called CEEBXITA) is invoked very early during the initialization process before the enclave initialization is complete. This exit is written in Assembler and not an LE/370-conforming HLL because the HLL environment is not established when this exit is invoked.

Early invocation of the assembler exit allows the enclave initialization code to benefit from any changes that might be contained in the exit. If run-time options are provided in the assembler exit, the enclave initialization code is aware of the new options. For example, you can use the assembler user exit to set the initial values of stack and heap storage using the STACK and HEAP run-time option settings.

The HLL initialization exit (CEEBINT) is invoked just before the invocation of the application code. In Language Environment/370 Version 1, this exit can be written in C/370 or in LE/370-conforming Assembler. The HLL initialization exit *cannot* be written in COBOL/370, although COBOL applications can use this HLL user exit. At the time when CEEBINT is invoked, the run-time environment is fully operational and all LE/370-conforming HLLs are supported.

You have the option of using either the assembler user exit or the HLL user exit for initialization for any application that you plan to run. Note, however, that the assembler and HLL exits do not perform exactly the same functions (see Chapter 27, "Advanced User Exit Topics" on page 173 for a detailed description of each exit).

If you modify either the assembler or HLL user exits, you must link them with your application in order to use them. If a modified HLL or assembler initialization user exit is not linked with your application, then the IBM-supplied defaults are used when your application runs.

User Exit for Termination

The CEEBXITA assembler user exit is invoked for enclave termination processing after all application code in the enclave has completed, but prior to any enclave termination activity.

CEEBXITA is invoked again when the LE/370 process terminates.

During termination, CEEBXITA can interrogate the LE/370 reason and return codes and if necessary, request an ABEND with or without a dump. This may be done at either enclave or process termination.

LE/370 does *not* provide a corresponding HLL termination user exit.

For more information about the user exits described in this chapter, including information about how to modify a user exit, see Chapter 27, "Advanced User Exit Topics" on page 173.

Invoking User Exits for Nested Enclave Initialization and Termination: CEEBXITA may also be invoked for initialization and termination of nested enclaves. CEEBXITA can differentiate easily between first and nested enclaves. For more information, see "Using the Assembler User Exit to Tailor the Environment" on page 173.

Linking and Running Your Program

This section describes how to link-edit and run your LE/370-conforming application under CMS, MVS, and TSO. The link-edit and run steps are generally the same for all LE/370-conforming HLL applications on a given platform, although there are some language-specific differences. If there is specific information for a language that you need to consider in order to complete one of the link-edit or run steps, you can find it at the end of the section that describes the step. The information is described language-by-language under the **Considerations** heading at the end of the chapter.

The following topics are discussed in this section:

Chapter 6. LE/370 Library Routine Considerations 27

Chapter 7. Making Your Application Reentrant 28
Advantages and Limitations of Reentrant Routines 28
Making Your COBOL Routine Reentrant 29
Making Your C/370 Program Reentrant 29

Chapter 8. Link-editing with LE/370 30
Link-editing Single-Language Applications 30
Link-editing ILC Applications 30

Chapter 9. Using the Linkage-Editor Under MVS 31
Providing Input to the Linkage-Editor 31
Writing JCL for the Linkage Editor 31
DD Statements for the Standard Data Sets 32
Linkage Editor Options 35
Detecting Link-Edit Errors 36

Chapter 10. Using the Loader under MVS 37
Providing Input to the Loader 37
Writing JCL for the Loader 38

Chapter 11. Running Your Application under MVS 41
Using the EXEC Statement 41
Specifying Run-time Options 42

Chapter 12. Creating a Load Module under TSO 44
Link-editing Your Application Using the LINK Command 44
Loading and Running a Load Module Using the LOADGO Command 45
Allocating Data Sets 47

Chapter 13. Running Your Application under TSO 48
Using the CALL Command 48
Allocating Data Sets 49
TSO Parameter List Format 49

Chapter 14. Link-editing under VM/CMS 50
Using the GLOBAL Command 50
Search Order for Dynamic Routines 51

© Copyright IBM Corp. 1991

Using the LOAD Command 51
Using the GENMOD command 53
Using the LKED Command 54

Chapter 15. Running Your Application under CMS 56
Using the START command 56
CMS Extended Parameter List 58
Using the C/370 CMOD EXEC 59
Using the C/370 LINKLOAD EXEC 61

Chapter 16. Using IBM-supplied Cataloged Procedures 62
Invoking Cataloged Procedures 62
IBM-supplied Cataloged Procedures 63
CEEWLG — Link and Run a Program Written in an LE/370-conforming HLL 64
CEEWL — Link an Application Written in an LE/370-conforming HLL 65
CEEWG — Load and Run a Program Written in an LE/370-conforming HLL 65
EDCPL — Prelink and Link a C/370 Application 66
EDCLIB — Invoke the Object Library Facility 68
Modifying Cataloged Procedures 70
Overriding and Adding to EXEC Statements 70
Overriding and Adding DD Statements 70

Chapter 6. LE/370 Library Routine Considerations

LE/370 library routines are broken into two categories: *resident* routines and *dynamic* routines. The resident routines are linked with the application and include such things as initialization/termination routines and *callable service stubs*. The callable service stubs contain addressing code to access the actual LE/370 callable service routines. The dynamic routines are not part of the application and are dynamically loaded during run-time.

The way LE/370 code is packaged keeps the size of application load modules small. In addition, maintenance of the dynamic library code does not require re-link editing of the application code.

VM Considerations: LE/370 callable service stubs and other routines such as those for initialization and termination are located in a text library called SCEELKED. LE/370 dynamically loaded library routines can be installed and supported in any of the following places under CMS:

- in a load library called SCEERUN
- as *relocatable load modules*
- as named shared segments (NSS)
- as nucleus extensions.

See the *Language Environment/370 Planning for Installation and Customization* guide for more information about LE/370 installation and support.

MVS Considerations: LE/370 callable service stubs and other routines such as those for initialization and termination are located in a library called SCEELKED. LE/370 dynamically loaded library routines are located in a library called SCEERUN. The LE/370 resident and dynamic libraries under MVS are located in MVS data sets identified with a high-level qualifier. For example, the data set containing the resident library might be called CEE.V1R1M0.SCEELKED, and the library containing dynamic routines might be called CEE.V1R1M0.SCEERUN.

COBOL Considerations: In the past (with OS/VS COBOL and VS COBOL II), it was possible to create applications that were self-contained load modules linked together with the library routines required to run the application. Under LE/370, however, most library routines (for example, callable services) *cannot* be statically linked to an application. Under LE/370, it is not possible to make a complete, self-contained load module.

COBOL Static and Dynamic Calls: COBOL allows you to specify at compile time whether your application will use *static* or *dynamic* calls to other subroutines, including LE/370 callable services.

- When a COBOL application makes a *static* call to another routine, the external references are resolved at link-edit time. When an LE/370 callable service is statically called, the callable service stub is link-edited with the application.
- When a COBOL application makes a *dynamic* call to another routine, the reference is resolved at run-time. When an LE/370 callable service is dynamically called, the callable service stub is not link-edited with the application.

For more information about static and dynamic calls, see the *IBM SAA AD/Cycle COBOL/370 Programming Guide.*

© Copyright IBM Corp. 1991

Chapter 7. Making Your Application Reentrant

Reentrancy allows more than one user to share a single copy of a load module.

The following applications *must* be reentrant:

- Routines to be used with CICS
- Routines to be pre-loaded with IMS/VS or IMS/ESA*
- Routines to be executed above 16 megabytes (however, note that C/370 routines automatically default to being above the 16 megabyte line).

C/370 Considerations: A reentrant routine is split into two parts:

- A part that contains external data
- A part that contains executable code and constant data.

Each user running the routine receives a private copy of the first (variable or nonreentrant) part. The second (constant or reentrant) part may be shared across multiple spaces or sessions.

External data is defined as data that persists over the lifetime of an enclave and maintains last-used values whenever a routine is entered.[2] Any routine in the enclave can refer to external data. This is unlike local data that is recognized only by the routine where it is defined.

Advantages and Limitations of Reentrant Routines

It is advantageous to make your routine reentrant if there will be multiple concurrent uses of the routine and if the routine is large. Less storage is used if there are many users who share the routine concurrently. Reentrancy also offers some performance enhancement because there is less paging to auxiliary storage.

C/370 Considerations: There are some limits to the advantages offered by reentrancy, however.

- Reentrancy requires an extra preparation step if your routine is written in C/370 and contains any external data. These routines must be run through the LE/370 prelink facility (see "Making Your C/370 Program Reentrant" on page 29).
- You cannot run an object module through the prelinker more than once.
- The shared portion of the routine must be installed in the MVS *link pack area* (LPA) or in the VM *named shared segment* (NSS) of your operating system. Installing a module in the LPA or NSS requires an initial program load (IPL) of your operating system.

[2] External data is called *writeable static* in the C language. The term *external data* will be used exclusively throughout in this book.

© Copyright IBM Corp. 1991

Making Your COBOL Routine Reentrant

If you intend to have multiple users execute a COBOL/370 routine (that is, program) at the same time, make your routine reentrant by specifying the RENT option when you compile it. For information about specifying the RENT compiler option, see the *IBM SAA AD/Cycle COBOL/370 Programming Guide.*

Making Your C/370 Program Reentrant

Under C/370, reentrant routines can be categorized by their *reentrancy type*, as follows:

Natural reentrancy Routines that contain no external data and do not require additional processing to make them reentrant

Constructed reentrancy Routines that contain external data and require additional processing to make them reentrant.

If your routine contains external data, use the LE/370 prelink facility to make your routine reentrant. This utility concatenates compile-time initialization information from one or more object modules into a single initialization unit. In the process, the external data part is mapped. If your routine does not contain any external data, you do not need to use the LE/370 prelink utility.

To generate a reentrant load module, you must follow these steps:

1. If your routine contains no external data, compile your routine as you would normally (no special compile-time options are required), and then go directly to step 4.
2. If your routine contains external data, you must compile your C source files using the `RENT` compile-time option.
3. Use the prelink utility to combine all input object modules into a single object module.

   Note: The output object module cannot be used as further input to the prelink utility.
4. Optionally, do **one** of the following:
   - Under VM, have your system programmer link/install your routine into an NSS of the system.
   - Under VM install your routine as a nucleus extension by using the VM `NUCXLOAD` command. For details about using VM commands, refer to the *VM/ESA CMS User's Guide* listed in "Bibliography" on page 477.
   - Under MVS, link the routine in the usual manner and have your system programmer install the routine in the LPA of the system.

For information about using the C/370 Prelink Facility, see Appendix A, "Pre-linking Your C Application" on page 431.

Chapter 8. Link-editing with LE/370

The linkage editor converts an object module into a load module and stores it in a program library. The load module becomes a permanent member of that library and can be retrieved at any time to run in the job that created it, or any other job.

Input to the linkage editor can include object modules, load modules, and control statements that specify how the input is to be processed. The output from the linkage editor can be a single load module or multiple load modules (using the NAME linkage editor control statement).

Link-editing Single-Language Applications

The entry point for an application is determined in one of two ways:

- The order of the routines presented to the linkage editor.
- Explicit specification of the entry point by link-edit control cards.

Table 7 identifies the default entry points for HLLs. The main routine entry point in an application is nominated by the END text record in the object deck produced by the compiler.

Table 7. Link-Edit Default Entry Point by Language

Language	Default Entry Point
C/370	CEESTART
COBOL	The name of the first object module presented to the linkage editor

Link-editing ILC Applications

When mixing languages within an application, present the desired main routine to the linkage editor first to nominate it as a main routine. In addition, use only one main routine to avoid ambiguity and undesired results.

Note: In order to take advantage of LE/370 ILC support, a re-link is required for existing ILC applications. The re-link of the application replaces the old library routines with the new LE/370 library routines.

For more information about link-editing ILC applications, including instructions about how to designate a main routine in each LE/370-conforming HLL, see "Determining the Main Routine" on page 143.

© Copyright IBM Corp. 1991

Chapter 9. Using the Linkage-Editor Under MVS

This section provides an overview of link-editing under MVS. For detailed information about link-editing, refer to the *Linkage Editor and Loader* publication listed in "Bibliography" on page 477.

Providing Input to the Linkage-Editor

Figure 20 shows the basic MVS link-editing process for your application.

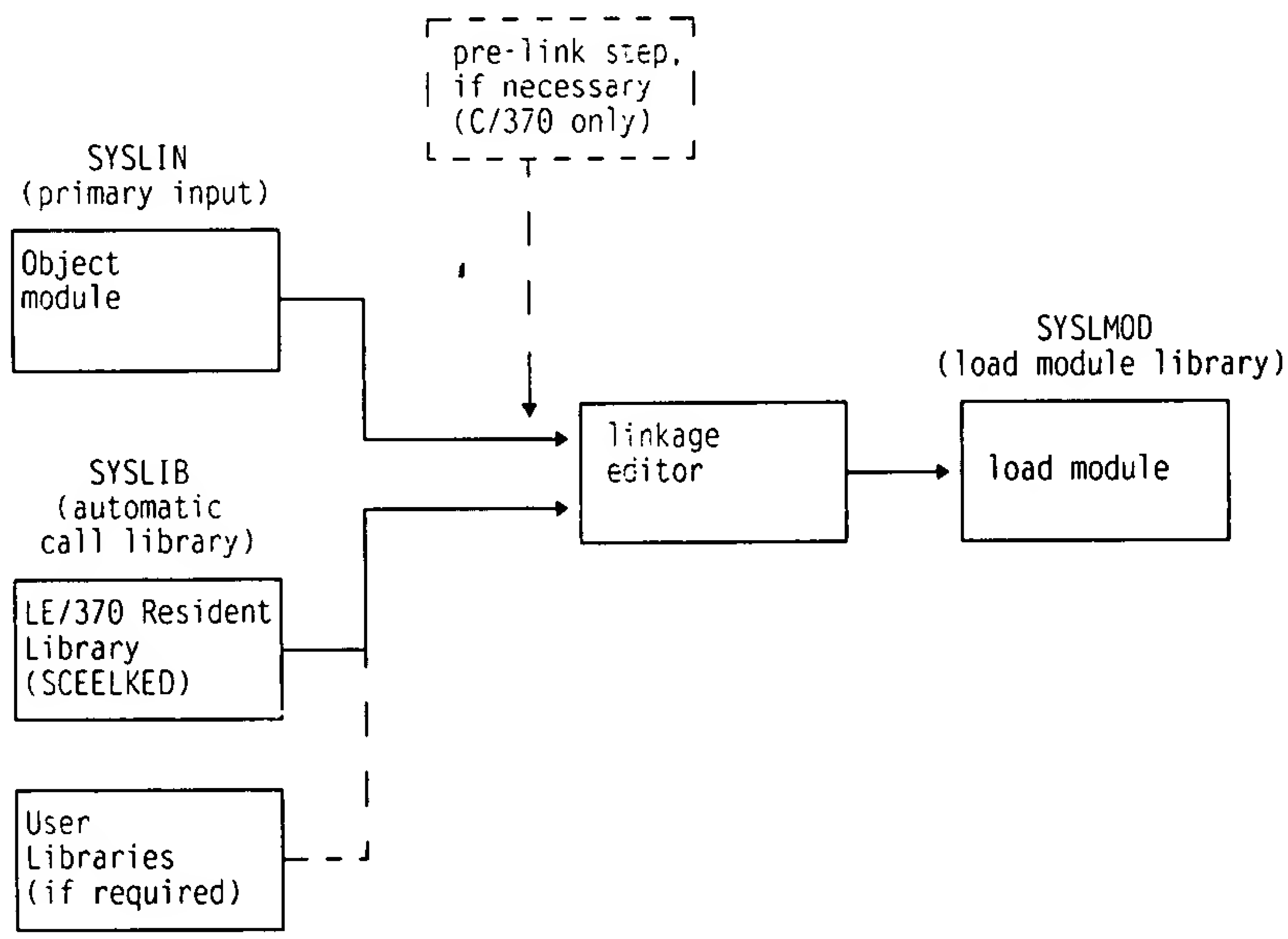


Figure 20. Basic Linkage Editor Processing

Input to the linkage-editor can be:

- One or more object modules
- Linkage editor control statements
- Previously link-edited load modules that you want to combine into a single load module.

Writing JCL for the Linkage Editor

Although you can use cataloged procedures rather than supply all of the job control language (*JCL*) required for a job step that invokes the linkage editor, you should be familiar with these JCL statements. This will enable you to make the best use of the linkage editor and, if necessary, override the statements of the cataloged procedure.

For a description of the IBM-supplied cataloged procedures that include a link-edit step, see Chapter 16, "Using IBM-supplied Cataloged Procedures" on page 62.

The following sections describe the basic JCL statements for link-editing.

© Copyright IBM Corp. 1991

Using the EXEC statement: Use the EXEC job control statement in your JCL to invoke the linkage editor. The EXEC statement to invoke the linkage editor is:

```
//LKED EXEC PGM=HEWL
```

Using the PARM parameter: Use the PARM parameter of the EXEC statement in your JCL for link-edit processing to specify link-edit options. For example, if you want a mapping of the load modules produced by the linkage editor, you specify the following:

```
//LKED EXEC PGM=HEWL,PARM='MAP'
```

For a description of link-edit options, see "Linkage Editor Options" on page 35.

DD Statements for the Standard Data Sets

The linkage editor always requires four standard data sets. You must define these data sets in DD statements with the ddnames SYSLIN, SYSLMOD, SYSUT1, and SYSPRINT.

A fifth data set, defined by a DD statement with the name SYSLIB, is necessary if you want to use the automatic call library. The five data set names and their characteristics are shown in Table 8.

Table 8. Data Sets Used for Link-Editing

ddname	Type	Function
SYSLIN[1]	Input	Primary input data, normally the output of the compiler
SYSPRINT[1]	Output	Diagnostic messages Informative messages Module map Cross-reference list
SYSLMOD[1]	Output	Output data set for the load module
SYSUT1[1]	Utility	Work data set
SYSLIB[2]	Library	Automatic call library
User-specified[3]		Additional object modules and load modules

Notes:

1 Required data set
2 Required for library run-time routines
3 Optional data set

Primary Input (SYSLIN): As indicated in Table 8, primary input to the linkage editor consists of a sequential data set, a member of a partitioned data set, or an in-line object module. The primary input must be comprised of one or more separately compiled object modules and/or linkage editor control statements. A load module cannot be part of the primary input, although it can be introduced by the control statement INCLUDE (see "Using the INCLUDE statement" on page 34).

Secondary Input (SYSLIB): Secondary input to the linkage editor consists of object modules or load modules that are not part of the primary input data set, but are to be included in the load module from the *automatic call library*. The automatic call library contains load modules or object modules that are to be used as

secondary input to the linkage editor to resolve external symbols left undefined after all the primary input has been processed.

The automatic call library can include:

- Libraries containing object modules, with or without linkage editor control statements
- Libraries containing load modules
- The library containing the LE/370 resident routines (SCEELKED).

SYSLIB is input to the linkage editor *only* if the NCAL link-edit option is not in effect (see Table 9 on page 35 or your *Linkage Editor and Loader* publication for more information).

You can also identify secondary input to the linkage editor with the INCLUDE statement. Use the INCLUDE statement to specify modules that you want to have included as secondary input (see "Using the INCLUDE statement" on page 34).

A routine compiled with an LE/370-conforming compiler cannot be executed until the appropriate LE/370 resident routines have been linked into the load module. The LE/370 resident routines include initialization/termination routines and LE/370 callable service stubs. The LE/370 resident library is contained in the SCEELKED library. Note that SCEELKED is installed under MVS into a data set with a high-level qualifier; for example, the data set name might be CEE.V1R1M0.SCEELKED. This data set must be specified in the SYSLIB statement in your JCL. If you are unsure of the name of the data set where SCEELKED has been installed at your location, contact your system administrator.

In the following example, the SYSLIB DD statement is written so that LE/370 resident library routines are included as secondary input into your load module:

```
//SYSLIB DD DSNAME=CEE.V1R1M0.SCEELKED,DISP=SHR
```

Output (SYSLMOD): Output (one or more load modules) from the linkage editor is always stored in a partitioned data set defined by the DD statement with the name SYSLMOD unless you specify otherwise. This data set is usually called a library.

Temporary Workspace (SYSUT1): The linkage editor requires a data set for use as a temporary workspace. It is defined by a DD statement with the name SYSUT1. This data set must be on a direct access device.

Listing (SYSPRINT): The linkage editor generates a listing that includes reference tables related to the load modules that it produces. You must define the data set where you want the linkage editor to store its listing in a DD statement with the name SYSPRINT.

Example of Linkage Editor JCL

A typical sequence of job control statements for link-editing an object module into a load module is shown in Figure 21. Note that the NAME linkage editor control statement in the figure will put PROGRAM1 in USER.LOADLIB with the member name PROGRAM1.

```
//LKED       EXEC  PGM=HEWL,PARM='MAP'
//SYSPRINT   DD    SYSOUT=A
//SYSLMOD    DD    DSNAME=USER.LOADLIB,UNIT=SYSDA,
//                 DISP=(NEW,KEEP),SPACE=(CYL,(10,10,1))
//SYSLIB     DD    DSNAME=CEE.V1R1M0.SCEELKED,DISP=SHR
//SYSLIN     DD    DSNAME=USER.OBJLIB(PROGRAM1),DISP=SHR
//           DD    DDNAME=SYSIN
//SYSUT1     DD    UNIT=SYSDA
//SYSIN      DD    *
     NAME  PROGRAM1(R)
/*
```

Figure 21. Creating a Load Module Under MVS Batch

Using the INCLUDE statement

Under MVS, you can use the INCLUDE linkage editor control statements to specify additional object or load modules that you want included in the output load module. Figure 22 contains an example of how to link-edit the CEEUOPT csect with your application.

Note: CEEUOPT is used to establish application run-time option defaults — see Chapter 30, "Specifying Run-time Options" on page 206 for more information.

```
//SYSLIN     DD    DSNAME=USER.OBJLIB(PROGRAM1),DISP=SHR
//           DD    DDNAME=SYSIN
//SYSIN      DD    *
     INCLUDE SYSLIB(CEEUOPT)

:
/*
```

Figure 22. Using the INCLUDE Linkage Editor Control Statement

Using the LIBRARY statement

Use the LIBRARY statement to specify additional libraries to be searched for object modules or load modules to be included in the load module. This statement has the effect of concatenating any specified member names with the automatic call library.

In Figure 23 on page 35, the LIBRARY statement is used to resolve the external reference PROGRAM2 from the library described on the TESTLIB DD statement.

```
//SYSLIN     DD    DSNAME=USER.OBJLIB(PROGRAM1),DISP=SHR
//           DD    DDNAME=SYSIN
//TESTLIB    DD    DSNAME=USER.TESTLIB,DISP=SHR
//SYSIN      DD    *
     LIBRARY TESTLIB(PROGRAM2)

:
/*
```

Figure 23. Using the LIBRARY Linkage Editor Control Statement

Data sets specified by the INCLUDE statement are incorporated as the linkage editor encounters the statement. In contrast, data sets specified by the LIBRARY statement are used only when there are unresolved references after all the other input is processed.

Linkage Editor Options

SYSLMOD and SYSPRINT are the data sets used for linkage editor output. The output from the linkage editor varies, depending on the options you select, as shown in Table 9.

Table 9. Selected Link-edit Options

Option	Function
XREF|NOXREF	Specifies whether a cross-reference list of data variables is to be generated
LIST|NOLIST	Specifies whether a listing of the linkage editor control statements is to be generated
NCAL	Specifies that the automatic library call mechanism should not be used to locate the modules referred to by the load module being processed. Use the NCAL command to suppress resolution of external differences. If you do not specify NCAL, the automatic call library mechanism will be used to locate the modules referred to by the load module being processed. Do not use NCAL if your application calls external routines that need to be resolved by automatic library call.
PRINT|NOPRINT	Specifies whether or not link-edit messages are to be written on the data set defined by the SYSLOUT DD statement.
MAP|NOMAP	Specifies whether a map of the load modules is to be generated and placed in the PRINT data set.

The PRINT default setting shown in Table 9 indicates that you always receive diagnostic and informative messages as the result of link-editing, even if you do not specify any options. You can get the other output items by specifying options in the PARM parameter in the EXEC statement in your JCL for link-editing. See “Using the PARM parameter” on page 32 and “Using the EXEC statement” on page 32 for more information.

For more information about link-edit options, see the *Linkage Editor and Loader* publication listed in “Bibliography” on page 477.

Detecting Link-Edit Errors

SYSPRINT produces a listing of diagnostic messages during link-editing. Check the load map to ensure that all the object and load modules that you expected were included.

When link-editing your application with the XREF link-edit option, you may get link-edit messages regarding unresolved "weak" external references. These messages do not necessarily indicate that there were errors when your application was link-edited.

You can find a description of link-edit messages in your *Linkage Editor and Loader* publication.

Chapter 10. Using the Loader under MVS

The loader processes object modules, loads the processed output into main storage, and executes it immediately. The loader is an all-in-one link-edit and execute step. It always stores the load modules directly in main storage where they are executed immediately. The load module is never written to disk. Once the load module has been executed, it cannot be used again without invoking the loader again.

Providing Input to the Loader

Your input to the loader can be:

- One or more object modules
- One or more previously link-edited load modules that you want to combine into a single load module
- A combination of both.

Unlike the linkage editor, you cannot use any control statements such as LIBRARY or INCLUDE with the loader. If any linkage control statements are used, they will be ignored. If you have asked for a list of error messages by including a DD statement in your JCL with the name SYSLOUT (see Table 10 on page 39), an informative error message indicating that the control statements are present in your JCL is printed on your listing.

In basic processing, as shown in Figure 24 on page 38, the loader accepts data from its primary input source, a data set defined by the DD statement with the name SYSLIN. This data is the object module produced by the compiler. The loader uses the external symbol dictionary in this object module to determine whether the module includes any external references that have no corresponding external symbols in the module.

© Copyright IBM Corp. 1991

As shown in Figure 24, the loader searches the automatic call library (SYSLIB) for the routines in which the external symbols are defined and includes them in the load module if they exist. If all external references are resolved satisfactorily, the load module is executed.

Your application cannot be executed until the appropriate run-time routines have been included. All run-time routines are included during loading.

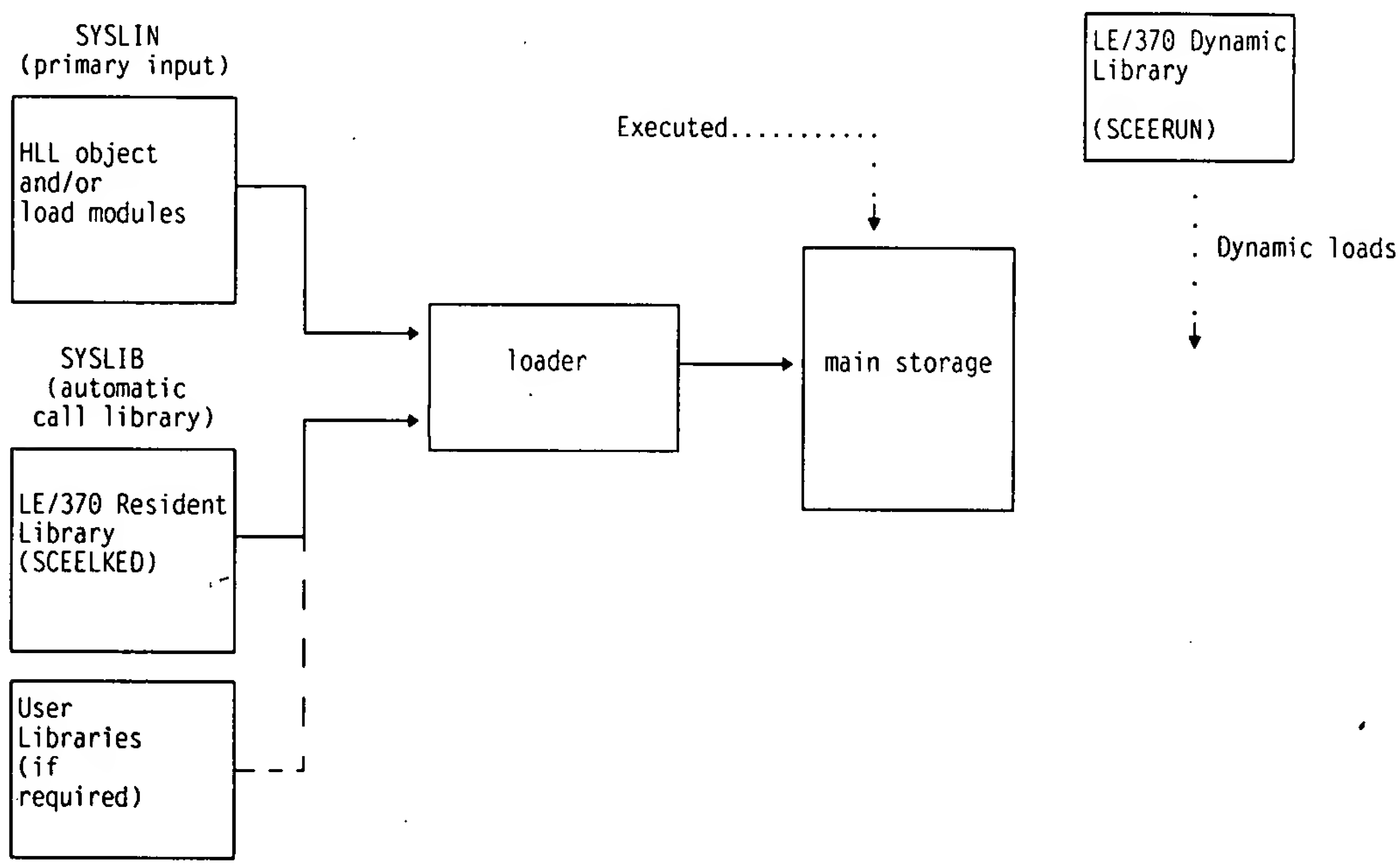


Figure 24. Basic Loader Processing

Writing JCL for the Loader

Although you will probably use cataloged procedures rather than supply all of the JCL required for a job step that invokes the loader, you should be familiar with these JCL statements so that you can make the best use of the loader and, if necessary, override the statements of the cataloged procedure.

For a description of the IBM-supplied cataloged procedures that include a loader step, see Chapter 16, "Using IBM-supplied Cataloged Procedures" on page 62.

The following paragraphs describe the basic JCL statements for loading.

Using the EXEC statement: Use the EXEC statement to invoke the loader. The EXEC statement to invoke the loader is:

```
//GO EXEC PGM=LOADER
```

Using the PARM parameter: In your JCL for loader processing, use the PARM parameter of the EXEC statement to specify loader options. For example, if you want your application to run even if abnormal conditions are detected, and you want a mapping of the load module, specify the following:

```
//GO EXEC  PGM=LOADER,PARM='MAP,LET'
```

See Table 11 on page 40 for a description of loader options.

DD Statements for the Standard Data Sets

The loader always requires one standard data set, that defined by the DD statement with the name SYSLIN. Three other standard data sets are optional and if you use them you must define them in DD statements with the names SYSLOUT, SYSPRINT, and SYSLIB. The four data set names and characteristics are shown in Table 10.

Table 10. Loader Standard Data Sets

DDNAME	Type	Function
SYSLIN	Input	Primary input data (normally the compiler output)
SYSLOUT	Output	Loader messages and module map listing
SYSPRINT	Output	Run-time messages and problem output listing
SYSLIB	Library	Automatic call library

Using the Loader under MVS to Run an Application

The sample in Figure 25 shows the general job control procedure for creating and running a load module under MVS.

```
//STEP1      EXEC   PGM=LOADER,PARM='MAP,LET'
//STEPLIB    DD     DSN=CEE.V1R1M0.SCEERUN,DISP=SHR
//SYSLIB     DD     DSN=CEE.V1R1M0.SCEELKED,DISP=SHR
//           DD     DSN=USER.LOADLIB,DISP=SHR
//SYSLIN     DD     DSN=USER.OBJLIB(PROGRAM1),DISP=SHR
//SYSLOUT    DD     SYSOUT=A
//SYSPRINT   DD     SYSOUT=A

  :
/*
```

Figure 25. JCL for Creating a Load Module

Requesting Loader Options

When you run the loader, you can request the options in Table 11 using the PARM parameter of the EXEC statement (see the *Linkage Editor and Loader* manual listed in "Bibliography" on page 477 for more information about specifying and using loader options).

Table 11. Selected Loader Options

Option	Function
MAP|NOMAP	Specifies whether a map of the load module is to be produced on SYSPRINT, giving the length and location of the main routine and all subroutines.
LET|NOLET	Specifies whether the loader will allow the load module to run, even when abnormal conditions have been detected.
CALL|NOCALL	Specifies whether or not the loader will attempt to resolve external references.
EP=*name*	Specifies the name of the entry point of the application being loaded.
PRINT|NOPRINT	Specifies whether loader messages will be listed in the data set defined by the SYSLOUT DD statement.
RES|NORES	Specifies whether the link pack area should be searched to resolve external references.
SIZE=*size*	Specifies the amount of storage to be allocated by loader processing; *size* includes the size of your load module.

Chapter 11. Running Your Application under MVS

Under MVS, you process an application by submitting batch jobs to the MVS operating system. A *job* may consist of one or more of the following job steps:

1. Compiling your application
2. Link-editing your application
3. Running your application.

IBM-supplied cataloged procedures are available that allow you to easily compile, link-edit or load, and/or run. For information about cataloged procedures, see Chapter 16, "Using IBM-supplied Cataloged Procedures" on page 62. If the statements in the cataloged procedures do not match your requirements exactly, you can modify them or add new statements for the duration of the job.

Using the EXEC Statement

Under MVS, you can request the execution of a load module in a EXEC statement in your JCL. The EXEC statement marks the beginning of each step in a job or procedure, and identifies to MVS the load module or cataloged procedure that MVS executes.

The general form of the EXEC statement is:

//[*stepname*] EXEC PGM=*program_name*

where *program_name* is the name of the member name or alias of the application that is to be executed. The specified application must be one of the following:

- A load module that is a member of a system library. Examples of system libraries are SYS1.LINKLIB and libraries specified in the LINKLIST.
- A load module that is member of a private library specified in a JOBLIB DD statement in your JCL
- A load module that is a member of a private library specified in a STEPLIB DD statement in your JCL.

Unless you have indicated that the load module is in a private library, MVS assumes that the load module is in a system library and searches the system libraries for the name you have specified.

You can define the library in a DD statement:

- with the ddname JOBLIB, immediately after the JOB statement in your JCL. This library will be searched before the system libraries. If any load module is not found in the JOBLIB, the system will look for it in the system libraries.

  In the following example, the system searches the private library USER.LOADLIB for the member PROGRAM1, reads the member into storage, and executes it:

  ```
  //JOB8     JOB  DAVE,MSGLEVEL=(2,0)
  //JOBLIB   DD   DSNAME=USER.LOADLIB,DISP=SHR
  //STEP1    EXEC PGM=PROGRAM1
  ```

- with the ddname STEPLIB at any point in the job step. The STEPLIB is searched before any system library or JOBLIB specified in a JOBLIB DD state-

© Copyright IBM Corp. 1991

ment for the *job step* in which it appears (although a load module can also be passed to subsequent job steps in the normal way).

The library search order is (in order of precedence):

1. Library specified in STEPLIB statement
2. Library specified in JOBLIB statement
3. The system library.

In the following example, the system searches USER.LOADLIB for the routine PROGRAM1 and USER.LOADLIB2 for the routine PROGRAMA:

```
//JOB8     JOB  DAVE,MSGLEVEL=(2,0)
//STEP1    EXEC PGM=PROGRAM1
//STEPLIB DD   DSNAME=USER.LOADLIB,DISP=SHR
//*
//STEP2    EXEC PGM=PROGRAMA
//STEPLIB DD   DSNAME=USER.LOADLIB2,DISP=SHR
```

Specifying Run-time Options

Each time your application runs, a set of run-time options must be established. These options determine many of the properties of the application's execution, including its performance, error handling characteristics, storage management, and production of debugging information. Under MVS, you can specify run-time options in any of the following places:

- In the CEEDOPT CSECT, where the installation default options are located (see Chapter 30, "Specifying Run-time Options" on page 206 for more information).
- In the CEEUOPT CSECT where user-supplied default options are located (see Chapter 30, "Specifying Run-time Options" on page 206 for more information).
- `#pragma runopts` in C source code (see Chapter 30, "Specifying Run-time Options" on page 206 for more information)
- In the PARM parameter of the EXEC statement in your JCL (see below).
- On the GPARM parameter of the IBM-supplied cataloged procedure (see Chapter 16, "Using IBM-supplied Cataloged Procedures" on page 62 for more information).
- In the assembler user exit (see Chapter 27, "Advanced User Exit Topics" on page 173 for more information).

Specifying Run-time Options in the EXEC Statement

The general form for specifying run-time options in the PARM parameter of the EXEC statement is:

```
//[stepname] EXEC PGM=program_name,
//              PARM='[run-time options/][program parameters]'
```

For example, if you want to generate a storage report and run-time options report for the application PROGRAM1, specify the following:

```
//GO1    EXEC PGM=PROGRAM1,PARM='RPTSTG(ON),RPTOPTS(ON)/'
```

Note that the run-time options that are passed to the main routine are followed by a slash (/) to separate them from program parameters.

C/370 Considerations: If you are a C/370 user, and EXECOPS is in effect (this is the default), you can pass run-time options in the EXEC statement in your JCL.

If NOEXECOPS is in effect, any run-time options specified in the EXEC statement will be treated as program parameters. When NOEXECOPS is specified, a slash is not required to precede program parameters in the EXEC statement.

See "C compatibility considerations" on page 208 for more information.

COBOL considerations: Under the VS COBOL II run-time, the order of run-time options and program parameters is the following:

```
program parameters/run-time options
```

As an aid to migration, LE/370 provides the CBLOPTS run-time option (see Chapter 30, "Specifying Run-time Options" on page 206 and "CBLOPTS" on page 222). If CBLOPTS(ON) is specified and the main routine is COBOL, the options on the command line will be honored in the same order as in the VS COBOL II run-time.

CBLOPTS(ON) must be specified as a default run-time option in CEEDOPT or CEEUOPT in order to provide compatibility with LE/370. CBLOPTS *cannot* be specified on application invocation.

Chapter 12. Creating a Load Module under TSO

Under TSO, you process an application by performing the following steps:

1. Compiling your application
2. Link-editing your application
3. Running your application.

The compiler produces an object module on a data set. The linkage editor processes the object module and readies it for execution. To link-edit your application under TSO, use the LINK command to create a load module from one or more object modules (plus any needed LE/370 library modules). To run your application, use the CALL command. Use the LOADGO command to link-edit and run your application in one step.

Link-editing Your Application Using the LINK Command

Use the LINK command to invoke the linkage editor. The linkage editor converts one or more object modules into a load module suitable for execution. You can run the load module later using the CALL command ("Using the CALL Command" on page 48). The general form of the LINK command is:

```
▶▶──LINK──(──data-set-list──)──┬──────────────────────────────┬──────────▶
                               └─LOAD──(──data-set-name──)────┘

▶──┬─────────────────────────────┬──┬───────────┬────────────────────────▶◀
   └─LIB──(──data-set-list──)────┘  └─options───┘
```

where

LINK (*data-set-list*)
: specifies the names of the data sets holding the object modules that you want to be link-edited. *data-set-list* must contain at least one object module, but can also contain linkage editor control statements. If you have only one name, you can omit the parentheses. If you have several names, you must separate them by commas or blanks within the parentheses.

LOAD (*data-set-name*)
: specifies the name of the data set that is to contain the load module generated by the linkage editor. If you specify a simple name, the system adds the user-identification qualifier and the descriptive qualifier LOAD (*userid.data-set-name*.LOAD) and uses that as the data set name. If you do not supply a member name, the load module is placed in *member* TEMPNAME of the *userid.data-set-name*.LOAD data set.

LIB (*data-set-list*)
: specifies the names of data sets that contain user-supplied modules that you want to be link-edited by the automatic library call facility.

  The LE/370 resident library, SCEELKED, must be included when specifying the *data-set-list* on the LIB option. SCEELKED is installed under MVS/TSO into a data set with a high-level qualifier; for example, the name might be CEE.V1R1M0.SCEELKED. If you are unsure of the name of the data set

© Copyright IBM Corp. 1991

where SCEELKED has been installed at your location, contact your system administrator.

options
specifies a list of linkage editor options. You must separate the options with a valid delimiter such as a comma or blank space.

Table 12 on page 47 contains a partial listing of link-edit options available under TSO.

The following example shows how to:

- link-edit two object modules named PROGRAM1 and CEEUOPT.

  Note: CEEUOPT can be used to establish programmer run-time option defaults — see Chapter 30, "Specifying Run-time Options" on page 206.

- specify the LE/370 library CEE.V1R1M0.SCEELKED as the automatic call library
- place the resulting load module in *member* PROGRAM1 in the library USER.LOADLIB
- direct the linkage editor listing to the terminal.

```
LINK ('USER.OBJLIB(PROGRAM1)','USER.OBJLIB(CEEUOPT)')
     LOAD('USER.LOADLIB(PROGRAM1)')
     LIB ('CEE.V1R1M0.SCEELKED') MAP PRINT(*)
```

For more information about using the TSO LINK command and its options, see the *OS/VS TSO Command Language Reference*.

Loading and Running a Load Module Using the LOADGO Command

Use the LOADGO command to create a load module in main storage and then run it. When the application has run, TSO automatically deletes the load module created by LOADGO. The general form of the LOADGO command is:

```
►►──LOADGO──(──data-set-list──)──┬──────────────────────────────┬──►
                                 └─'program-parameter-string'─┘

►──┬──────────────────────────┬──┬──────────┬──────────────────►◄
   └─LIB──(──data-set-list──)─┘  └─options──┘
```

where:

LOADGO (*data-set-list*)
specifies the names of one or more object modules and/or load modules that you want to be loaded and run. If you have only one name, you can omit the parentheses. If you have several names, they must be separated by commas or blanks.

The names can be data set names, names of members of data sets, or both.

program-parameter-string
specifies run-time options and program parameters that you want to pass to the load module at run-time. The run-time options and parameters that are passed

to the main routine in the load module are separated by a slash (/). The possible combinations are described in Chapter 30, "Specifying Run-time Options" on page 206.

LIB (*data-set-list*)
specifies the names of data sets that contain user-supplied modules that you want to be link-edited by the automatic library call facility.

You must also list the LE/370 resident library, SCEELKED. SCEELKED is installed under MVS/TSO into a data set with a high-level qualifier; for example, the name might be CEE.V1R1M0.SCEELKED. If you are unsure of the name of the data set where SCEELKED has been installed at your location, contact your system administrator.

options
specifies a list of loader options. You must separate the options with a valid delimiter such as a comma or blank space.

For a description of loader options, see Table 12 on page 47 as well as the *OS/VS TSO Command Language Reference* publication.

C/370 Considerations: If you are a C/370 user, and EXECOPS is specified (it is the default) you can pass run-time options in the LOADGO command.

If NOEXECOPS is in effect, any run-time options specified in the LOADGO command will be treated as program parameters. When EXECOPS is specified, a slash is not required to precede program parameters.

See "C compatibility considerations" on page 208 for more information.

COBOL Considerations: Under the VS COBOL II run-time, the order of run-time options and program parameters is the following:

```
program parameters/run-time options
```

If CBLOPTS(ON) is specified, the options on the command line will be honored in the VS COBOL II order.

See "COBOL considerations" on page 43 for more information.

The following example shows how to:

- Create a load module using the object modules PROGRAM1 and CEEUOPT
- Specify the run-time options to produce the storage report and run-time options report
- Specify the LE/370 library CEE.V1R1M0.SCEELKED as the automatic call library
- Generate a mapping of the load module that is run
- Direct the loader listing to the terminal.

```
LOADGO ('USER.OBJLIB(PROGRAM1)','USER.OBJLIB(CEEUOPT)')
       'RPTOPTS(ON),RPTSTG(ON)/'
        LIB ('CEE.V1R1M0.SCEELKED') MAP PRINT(*)
```

Note: In order to run an application successfully under TSO, the SCEERUN dynamic library must be either in the TSO logon procedure, or in the link list concatenation. See *Language Environment/370 Planning for Installation and Customization* for more information about the TSO logon procedure.

Table 12. Selected Loader Options

Option	Action
CALL|NOCALL	CALL specifies that the data set specified in the **LIB** operand is to be searched to locate load modules referred to by the module being processed. NOCALL specifies that the data set specified in the **LIB** operand does not search to locate load modules referred to by the module being processed.
SIZE(*integer*)	Specifies the size, in bytes, of the dynamic storage that can be used by the loader.
PRINT(*data_set_name*)	PRINT specifies the name of the data set that is used to contain the listing. You can direct output to the terminal by specifying PRINT(*).
MAP|NOMAP	MAP generates a map of the load modules and places them in the PRINT data set. NOMAP suppresses the map listing.
LET|NOLET	LET specifies that the loader will try to execute your application even if an error of severity 2 is found. NOLET suppresses execution if an error of severity 2 or greater is found.
RES|NORES	RES specifies that the link pack area is to be searched for the load module (referred to by the module being processed) before the specified libraries are searched. If you also specify the NOCALL operand, the RES option is invalid. NORES specifies that the link pack area is not to be searched for the load module referred to by the module being processed.

Allocating Data Sets

When you use the LOADGO command under TSO, you must also specify the ALLOCATE command to dynamically allocate the data sets required by the application you intend to run. For more information, see the *OS/VS TSO Command Language Reference* publication.

Chapter 13. Running Your Application under TSO

You can execute your TSO application in the following ways:

- Use LOADGO to create a module in main storage and then running it. For a description of the LOADGO command, see "Loading and Running a Load Module Using the LOADGO Command" on page 45.
- Use the CALL command to run a load module that you have created using LINK.

Using the CALL Command

The TSO CALL command loads and executes a specified load module.

The general form of the CALL command is:

```
 Syntax
▸▸──CALL──┬─────────────────────────────┬──┬─────────────────┬──────▸◂
          ├─data_set_name───────────────┤  └─program_parms───┘
          ├─data_set_name(membername)───┤
          └─(membername)────────────────┘
```

where

data-set-name
: specifies the partitioned data set member that holds the load module. If you specify the simple name of the data set, the system assumes the descriptive qualifier LOAD. If you do not specify a member name, the system assumes the name TEMPNAME.

  You can also specify the member name of the data set that holds the load module you plan to run, as indicated in the syntax diagram.

program_parms
: is a list of run-time options and program parameters passed to the main routine. Note that run-time options and program parameters are separated by a slash(/).

For example, if you wanted to load and run the load module PROGRAM1 located in the data set USER.LOADLIB, and pass run-time options that generate storage and run-time options reports, specify the following:

```
CALL 'USER.LOADLIB(PROGRAM1)' 'RPTSTG(ON),RPTOPTS(ON)/'
```

Note: In order to run an application successfully under TSO, the SCEERUN dynamic library must be either in the TSO logon procedure, or in the link list concatenation. For more information about the TSO logon procedure, see *Language Environment/370 Planning for Installation and Customization.*

C/370 Considerations

If you are a C/370 user, and the EXECOPS run-time option is specified (it is the default), you can pass run-time options in the LOADGO command.

© Copyright IBM Corp. 1991

If the NOEXECOPS run-time option is in effect, run-time options cannot be specified on the command line. A slash is not required to precede program parameters in the CALL command when NOEXECOPS is in effect. Note that any run-time options specified in the CALL command will be treated as program parameters. For more information, see "C compatibility considerations" on page 208.

For more information about specifying run-time options, see Chapter 30, "Specifying Run-time Options" on page 206.

COBOL Considerations: Under the VS COBOL II run-time, the order of run-time options and program parameters is the following:

```
program parameters/run-time options
```

If CBLOPTS(ON) is specified, the options on the command line will be honored in this order.

See "COBOL considerations" on page 43 for more information.

Allocating Data Sets

When you use the CALL command under TSO, you must also specify the ALLOCATE command to dynamically allocate the data sets required by the application you intend to run. For more information, see the *TSO/E Command Language Reference* publication listed in "Bibliography" on page 477.

TSO Parameter List Format

COBOL Considerations: Your COBOL/370 application (that is, an application with a COBOL main routine) receives the TSO parameter list as a halfword prefixed string.

C Considerations: When executing under TSO with the IBM-supplied default setting of PLIST(HOST), LE/370 dynamically determines whether or not a CPPL (command processor parameter list) has been passed. If so, your C/370 application (that is, an application with a C MAIN routine) receives the TSO parameter list as an `argc, argv` style format.

If PLIST(TSO) is in effect, the inbound parameter list is a CPPL pointed to by R1. The CPPL is accessible using the `__sysplist` macro as described in "C/370 Parameter Passing Styles" on page 14.

Chapter 14. Link-editing under VM/CMS

Compilation of your application under VM produces an object module with the file type TEXT. Before you run the application, external references inserted by the compiler must be resolved. You can use one of the following methods to create an executable application:

- **Create a temporary copy of your application** in virtual storage by using the LOAD, and possibly INCLUDE, commands. No permanent copy of the executable application is made.
- **Create a module** using the LOAD, possibly the INCLUDE, and the GENMOD commands. A module is an executable application that is stored as a file with a file type of MODULE on a CMS disk.
- **Create a module in a member of a library** using the LKED command. This method link-edits an executable application and stores it as a load module in a member of a CMS LOADLIB.
- **Create a module** using the CMOD exec (C/370 applications only). See "Using the C/370 CMOD EXEC" on page 59 for more information.

Your application can be executed after you complete any of the above steps.

Using the GLOBAL Command

Note that before you use the CMS LOAD command and before running applications under CMS, you must issue a GLOBAL command.

For example, before a CMS LOAD command is issued, a GLOBAL TXTLIB must be issued for the LE/370 text library to resolve external references to the LE/370 resident routines. For example, before *loading* your application, issue this command:

```
GLOBAL TXTLIB SCEELKED usertxt
```

where SCEELKED is the LE/370 text library and *usertxt* is the name of any user-generated text library or libraries that you want searched for other text files your application will need to run.

C/370 Considerations: If your C/370 application will perform long double arithmetic, you must also specify the CMSLIB text library in your GLOBAL TXTLIB command.

Before *running* your application, a GLOBAL LOADLIB must be issued for the LE/370 LOADLIB to resolve external references to the LE/370 dynamic routines.

For example, before *running* your application, issue this command:

```
GLOBAL LOADLIB SCEERUN userload
```

where SCEERUN is the LE/370 load library and *userload* is the name of any user-generated load library or libraries that you want searched for load modules that your application will need to run.

© Copyright IBM Corp. 1991

Note: Not all LE/370 dynamic routines are located in the SCEERUN LOADLIB. Some may be in relocatable load modules, a nucleus extension, or an NSS. Check with your system administrator to determine where the dynamic routines are located at your installation.

Search Order for Dynamic Routines

The search order for dynamically loaded routines is :

1. Nucleus extension
2. Saved Segments (formerly known as DCSS)
3. Relocatable load modules
4. Load modules in LOADLIBs
5. Object decks
6. TXTLIB members.

Normal search order prevails when searching for a particular type in the previous list. For example, files on the A-disk are searched before files on the B-disk.

Using the LOAD Command

The loader is invoked under VM by using the LOAD command. Use the LOAD command to read one or more TEXT files (containing relocatable object code) or members of a text library from a minidisk or directory and load them into virtual storage, thus establishing proper linkages between the files. The file containing the main routine should be the first file named in the command.

The general syntax of the LOAD command is:

LOAD *filename1 filename2 ...filenamen [(options*

where

filename
is the name of a file you want to load into storage.

options
is a list of LOAD options separated by blanks or commas (see Table 13 for a partial listing of available options)

For more information about LOAD and its options, see the *IBM VM/ESA* CMS System Command Reference* listed in "Bibliography" on page 477.

For example, the following example causes the text library containing LE/370 resident routines, SCEELKED, and the USERTXT text library to be searched for files that your application will need to run. The files PROGRAM1 and CEEUOPT are loaded into virtual storage and a load map is written:

```
GLOBAL TXTLIB SCEELKED USERTXT
LOAD PROGRAM1 CEEUOPT (MAP
```

Table 13. Selected CMS Load Options

Option	Function
RESET *entry*|*	RESET sets the starting location for the applications currently loaded. *entry* must be an external name (for example, a CSECT control section or ENTRY) in the loaded applications. If you specify *, the results are the same as if the RESET option were omitted. If the RESET option is omitted, the default entry point is used.
<u>MAP</u>|NOMAP	MAP writes a load map to a file in your minidisk or directory named LOAD MAP A5. If you specify NOMAP, no LOAD MAP file is created.
TYPE|<u>NOTYPE</u>	TYPE displays the load map at your terminal and writes it to a file on minidisk or directory. NOTYPE does not display the file at your terminal
<u>LIBE</u>|NOLIBE	LIBE searches text libraries for missing subroutines. The text libraries must be previously defined by a GLOBAL command. NOLIBE does not search text libraries for unresolved differences.
START	Executes the application when loading has completed.
<u>NORLDsav</u>|RLDsave	NORLDsav instructs the CMS LOADER not to save relocation information from the TEXT files being loaded. RLDsave instructs CMS to save relocation information from the text files. The GENMOD command uses relocation information to generate relocatable CMS modules.
AMODE 24|31|ANY	Specifies the addressing mode of the application in a 370-XA mode virtual machine. In a System/370 mode virtual machine, you may specify AMODE, although only 24-bit addressing is available. This allows you to create XA capable modules files on a System/370 mode virtual machine. Valid AMODE values are: • 24 - The entry point of the application receives control in 24-bit addressing mode. • 31 - The entry receives control in 31-bit addressing mode when running on a 370-XA mode virtual machine, and in 24-bit mode on a System/370 mode virtual machine. • ANY - The entry point is capable of operating in either 24- or 31-bit addressing mode.
RMODE 24|ANY	Specifies, in a 370-XA mode virtual machine with greater than 16M of storage, the location where the loaded application(s) is to reside. Valid RMODE values are: • 24 - The load module must reside below the 16M line in a 370-XA virtual mode machine. • ANY - The load module must reside above the 16M line in a 370-XA virtual mode machine.

C/370 Considerations: If the main routine is C/370, then under the options for the LOAD command specify `RESET CEESTART`.

Issuing LOAD and INCLUDE Commands

The LOAD command loads a TEXT file or member of a text library into virtual storage. Use the INCLUDE command to load additional TEXT files or members of a text library that are to comprise your executable application.

The INCLUDE and LOAD commands have similar formats and option lists. The main difference is that if you issue two LOAD commands in succession, the second command replaces the first. The INCLUDE command, on the other hand, cannot be used unless you have just issued a LOAD. You may specify as many INCLUDE commands as necessary following the LOAD command to load files into storage. The files specified in the INCLUDE command must refer to subroutines.

The general form of the INCLUDE command is

```
INCLUDE filename1, filename2, filenamen...[(options]
```

The following example loads a TEXT file from the USERTXT text library and INCLUDEs another TEXT file from another text library into the load module.

```
GLOBAL TXTLIB SCEELKED USERTXT USERTXT2
LOAD PROGRAM1
INCLUDE PROGRAM2 (MAP
```

Using the GENMOD command

Use the LOAD and INCLUDE commands in conjunction with the GENMOD command to create application modules. A module is a non-relocatable file whose external references have been resolved. In CMS, these files must have a file type of MODULE.

The general form of the GENMOD command is:

```
GENMOD filename  [(options]
```

The GENMOD command takes a copy of the executable module in virtual storage and stores it on disk with a *filename* that you specify. In the following example, PROGRAM1, PROGRAM2, and PROGRAM3 are TEXT files that you are putting into a module with a filename of PROGRAM1 and a filetype of MODULE:

```
GLOBAL SCEELKED USERTXT
LOAD PROGRAM1 PROGRAM2 PROGRAM3
GENMOD PROGRAM1
```

If you use the name of an existing module, the old module is replaced. If you do not specify in *filename* the name of the file where you want the load module to be stored, the GENMOD command processor defaults to the first entry point in the load map.

After you create the module with GENMOD, any time that you want to run the application composed of the source files PROGRAM1, PROGRAM2,PROGRAM3, simply enter:

```
PROGRAM1
```

and the module will execute.

Note that before execution, you must enter GLOBAL LOADLIB commands to identify the libraries that are searched if your application makes any references to external subroutines. See "Using the GLOBAL Command" on page 50 for more information.

Using FILEDEF to define input and output files

If PROGRAM1 requires input and/or output files, you must define these files using the FILEDEF command prior to executing the module. The FILEDEF command relates the ddname of the input or output file specified in your application with an I/O device. For example, if PROGRAM1 contains a ddname of an input file stored on disk as MYDATA INPUT A, issue the following command:

```
FILEDEF infile DISK MYDATA INPUT A
```

where *infile* is the ddname of the input file specified in MYPROG1.

For more information about the GENMOD and FILEDEF commands, refer to the see the *IBM VM/ESA CMS System Command Reference* listed in "Bibliography" on page 477.

Using the LKED Command

The LKED command is used to create a member of a CMS load library. CMS load libraries, like text libraries, are in CMS partitioned data set formats. Unlike text libraries which contain applications that in turn contain unresolved external references to other routines, load libraries contain applications with external references that have already been resolved, thus saving overhead every time the application is loaded.

Your TEXT file is input to the LKED command. If your application calls a subroutine with object code stored as a separate TEXT file or as a member of a text library, you must define the files that contain the subroutines used by your application with a FILEDEF command.

After you issue the appropriate FILEDEF commands, issue the LKED command as follows:

Syntax

```
▸▸──LKED──filename──┬──────────────────┬──┬─────────────┬──────▸◂
                    └──(NAME membname──┘  └─LIB libname─┘
```

where

filename is the filename of the TEXT file that contains your object code and or linkage editor control cards.

NAME membname is the member name to be used for the load module that is created.

LIBE libname is the filename of the LOADLIB file where the resulting load module is placed.

The following example:

- causes the automatic call library to search SCEELKED to resolve external references
- creates a load library member named PROGRAM1
- and stores it in a CMS load library with the name USERLOAD.

```
FILEDEF SYSLIB DISK SCEELKED TXTLIB E
LKED PROGRAM1 (NAME PROGRAM1 LIBE USERLOAD
```

For more information about the LKED and a complete list of options, see the *IBM VM/ESA CMS System Command Reference* listed in "Bibliography" on page 477.

Chapter 15. Running Your Application under CMS

You can run an application under CMS after

- Issuing the LOAD command to store a copy of it in virtual storage
- Issuing a GENMOD command to store it on disk
- Issuing the LKED command to store it in a LOADLIB.

Considerations for Dynamic ILC Calls: If your ILC application contains dynamic calls and you are running under CMS, your application must be stored in a CMS LOADLIB or a CMS relocatable load library.

Using the START command

After you load your application into virtual storage using the LOAD command and issue the appropriate GLOBAL commands, use the CMS START command to execute your application. The general format of the START command is shown below.

Syntax

```
▸▸——START—*—┬——————————————————————————┬——————————————————▸◂
            └—program_parameter_string—┘
```

where

* specifies that control passes to the application's default entry point at execution time (see Chapter 8, "Link-editing with LE/370" on page 30 for information about how the default entry point is determined). You can also indicate an entry point whose name you specify in the application.

program_parameter_string
run-time options and program parameters passed to the main routine in the application. C/370 users may omit the slash (/) if the NOEXECOPS run-time option is in effect.

In the following example, the compiled application PROGRAM1 is loaded and run with the RPTSTG(ON) and RPTOPTS(ON) run-time options specified:

```
GLOBAL TXTLIB SCEELKED USERTXT
LOAD PROGRAM1
GLOBAL LOADLIB SCEERUN
START * RPTSTG(ON),RPTOPTS(ON)/
```

In the case of a single application where you do not supply any run-time options or parameters, you can load and execute using a single command, the START option of the LOAD command:

```
LOAD PROGRAM1 ( START
```


©.Copyright IBM Corp. 1991

If PROGRAM1 requires input and/or output files, you must define these files using the FILEDEF command prior to executing the module. See "Using FILEDEF to define input and output files" on page 54 for more information.

C/370 Considerations: If you are a C/370 user, and EXECOPS is specified, you can pass run-time options in the START command or any other command in this chapter that allows you to pass run-time options. See "C compatibility considerations" on page 208 for more information.

COBOL Considerations: Under the VS COBOL II run-time, the order of run-time options and program parameters is the following:

```
program parameters/run-time options
```

If CBLOPTS(ON) is specified, the options on the command line will be honored in the VS COBOL II order. This applies to the START command and any other command in this chapter that allows you to pass run-time options.

See "COBOL considerations" on page 43 for more information.

Executing a Module Produced by the GENMOD Command

After you create a module using the GENMOD command, and you have issued the GLOBAL LOADLIB SCEERUN command, you can execute the module simply by entering the module name on the command line, as shown in the syntax below.

Syntax

```
▶▶──modname──┬────────────────────────────┬──────────────▶◀
             └─program_parameter_string─┘
```

The run-time options and parameters that are passed to the main routine are separated by a slash (/). Run-time options and program parameters are discussed in greater detail in Chapter 30, "Specifying Run-time Options" on page 206.

In the following example, PROGRAM1, PROGRAM2, and PROGRAM3 are TEXT files that you are putting into a module with a filename of PROGRAM1 and a filetype of MODULE. PROGRAM1 is executed by typing its name on the CMS command line with a list of run-time options that you want to pass to it:

```
GLOBAL TXTLIB SCEELKED USERTXT
LOAD PROGRAM1 PROGRAM2 PROGRAM3
GENMOD PROGRAM1
PROGRAM1 RPTSTG(ON),RPTOPTS(ON) /
```

Running a Module Produced by the LKED Command

After you create a module and store it in a LOADLIB using the LKED command, you can run it using the OSRUN command.

Before running your application you must issue a GLOBAL command to identify to CMS the LOADLIB containing the module, plus the LE/370 LOADLIB to identify LE/370 load modules that are called by your application:

```
GLOBAL LOADLIB SCEERUN userload
```

where SCEERUN is the name of the LE/370 load library, and *userload* is the name of the library containing the load module you want to execute.

The general format of the OSRUN command is shown below.

Syntax

```
▶▶──OSRUN──membname──┬────────────────────────────────┬──▶◀
                     └─PARM=──program_parameter_string─┘
```

where *membname* is the member name containing the load module that you created using LKED. The member name is in turn located in the load library that you identified in CMS using the GLOBAL command.

For example, if you wanted to run a routine named PROGRAM1, and you wanted to specify the RPTSTG(ON) and RPTOPTS(0N) run-time options, you would issue the following commands:

```
FILEDEF SYSLIB DISK SCEELKED TXTLIB E
LKED PROGRAM1 (NAME PROGRAM1 LIBE USERLOAD
GLOBAL LOADLIB SCEERUN USERLOAD
OSRUN PROGRAM1 PARM='RPTSTG(ON),RPTOPTS(ON)/'
```

CMS Extended Parameter List

When CMS transfers control to an application, a mechanism is established to obtain the arguments that are passed to the main routine. The CMS extended parameter list is used to construct the main routine's parameters and obtain run-time options. LE/370 repackages the CMS extended parameter list according to the format indicated in Chapter 3, "Parameter List Formats" on page 10. Under LE/370:

- Your COBOL/370 application (that is, an application with a COBOL main routine) receives the CMS extended parameter list as a halfword prefixed string
- Your C/370 application (that is, an application with a C main() routine) receives the CMS extended parameter list in an `argc, argv` style format.

Important: LE/370 does *not* honor the CMS tokenized parameter list format. If your application relies upon CMS commands that do not construct an extended parameter list (such as the CMS RUN command), unexpected results may occur.

Using the C/370 CMOD EXEC

The IBM-supplied `CMOD` exec invokes the loader, which loads one or more object modules into virtual storage, resolves external references, and creates an executable module with the filetype of TEXT. This EXEC can be used by C/370 applications only.

The general form of the CMOD EXEC is:

Syntax

```
          ┌──────────┐
          ↓          │
▸▸──CMOD────textdeck─┴──┬───────────────────────┬──────────▸◂
                        │     ┌──────────┐      │
                        │     ↓          │      │
                        └─(─────option───┴──────┘
```

where

textdeck
: is the name of the object module you want to run. Do not specify the file type or file mode when using this EXEC.

  Note: The file containing the main function should be the first named file.

options
: are the options you want to apply as the executable module is being generated. The options are listed in Table 14 on page 60.

Table 14. CMOD options

Option	Function
AUTO|NOAUTO	AUTO specifies that your disks are to be searched for TEXT decks to resolve undefined references. NOAUTO specifies that your disks are not to be searched for TEXT decks to resolve undefined references.
DUP|NODUP	DUP specifies that an error message will be generated if duplicate csect names are found. NODUP specifies that an error message will not be generated if duplicate csect names are found.
INV|NOINV	INV specifies that invalid card images are not to be included in the load map. NOINV specifies that invalid card images are to be included in the load map.
LET|**NOLET**	LET specifies that an attempt to generate the load module will be made even if there are load errors. NOLET specifies if there are load errors, no attempt to generate the load module will be made.
MAP|NOMAP	MAP specifies that a load map file is to be made and written to your A-disk with the name LOAD MAP A. NOMAP specifies that a load map file is not to be made.
MODNAME *modulename*	Specifies the name of the executable module. If you do not specify a *modulename*, the name of the first TEXT deck that you have specified will be used as the file name. The file type will be MODULE and the file mode will be A.
ORIGIN *address*	Specifies an address where the load module is to loaded. If you do not specify an *address*, the load module is loaded beginning at location 20000. **Note:** If you plan to use AD/Cycle CODE/370 in full screen mode, you must specify an ORIGIN >= 20000.
RLD|NORLD	RLD specifies that relocation directory information will be saved in the load module. NORLD specifies that relocation directory information will not be saved in the load module.
STR|NOSTR	STR specifies that storage is to be initialized during the generation of the load module. NOSTR specifies that storage will not be initialized during the generation of the load module.
AMODE	Specifies the addressing mode in which the application will be entered on an ESA* virtual machine.
RMODE	Specifies where the application will reside on an ESA virtual machine with greater than 16M of storage. For more information, see the *VM/ESA CMS Command Reference* listed in the "Bibliography" on page 477.

Using the C/370 LINKLOAD EXEC

The IBM-supplied `LINKLOAD` exec generates a fetchable member of a CMS load library.

The general form of the the LINKLOAD exec is

Syntax

```
            ┌──────────┐
            ↓          │
▸▸──LINKLOAD──textdeck──┴──(──LIB──libname──┬────────────────┬──▸◂
                                            │    ┌──────┐    │
                                            │    ↓      │    │
                                            └──(──option─┴────┘
```

where

textdeck
is the name of input text decks.

The file type of the text decks must be TEXT and the source code must contain a `#pragma linkage`(name,FETCHABLE) preprocessor directive.

Do not specify the file type or file mode when using this EXEC.

libname
specifies the name of the library where the member is to be stored.

options
are the options you want to apply as the fetchable load module is being generated. The options are listed in Table 15.

Table 15. LINKLOAD options

Option	Function
MBR	A keyword specifying that the next argument *memname*, is the name of the member within the load library that is to be generated. If you do not specify a *memname*, the name of the text deck containing the fetchable code is used.
ADD|REPLACE|NEW	*One* of these options may be specified on a given invocation of LINKLOAD: • ADD specifies that the load member generated by LINKLOAD is to be added to the load library. If a member by the same name already exists, the new member will not be added. • REPLACE specifies that the load member generated by LINKLOAD is to replace a member by the same name in the load library. If the member does not already exist, the new member will be added. • NEW specifies that an existing load library of the same name containing only the named member will be created.

Chapter 16. Using IBM-supplied Cataloged Procedures

A cataloged procedure is a set of job control statements usually stored in a system library, for example SYS1.PROCLIB. Note, however, that where cataloged procedures are stored is installation-defined and may differ at your location.

A cataloged procedure includes one or more EXEC statements, each of which can be followed by one or more DD statements. You can retrieve a cataloged procedure from the library by using its member name in an EXEC statement of a job statement in the input stream.

A cataloged procedure can contain statements for the processing of an entire job, or it can contain statements to process one or more steps of a job, with the remaining steps defined in job control statements in the input stream. A job can use several cataloged procedures, each processing one or more of the job steps. A job can also call for execution of the same cataloged procedure in more than one job step.

You can use cataloged procedures to save time and reduce JCL errors. If the statements in the procedure do not match your requirements exactly, you can modify them easily or add new statements for the duration of a job.

Note: The cataloged procedures shown in this section are intended for use as references and do not necessarily reflect the procedures as they are provided at your installation. If options are not explicitly supplied with the procedure, default options established at the installation apply. You can override these default options by using an EXEC statement that includes the desired options (see "Overriding and Adding to EXEC Statements" on page 70.).

Invoking Cataloged Procedures

To invoke a cataloged procedure, specify its name in the PROC parameter of an EXEC statement. You do not need to code the keyword PROC. For example, to use the cataloged procedure CEEWLG, include the following statement in an appropriate position among your other job control statements in the input stream:

```
//stepname    EXEC PROC=CEEWLG
```

or

```
//stepname    EXEC CEEWLG
```

Either of these EXEC statements could be used to call the IBM-supplied cataloged procedure CEEWLG to process the job step specified in *stepname*.

A job step that calls for execution of a cataloged procedure can also contain DD statements that are applicable to the job steps of the cataloged procedure. A job that calls for execution of a cataloged procedure may, in other steps:

- Call for execution of other cataloged procedures
- Call for other (single or multiple) executions of the same cataloged procedure
- Call directly for execution of load modules.

 © Copyright IBM Corp. 1991

IBM-supplied Cataloged Procedures

The IBM-supplied MVS cataloged procedures that you can use under LE/370 are listed in Table 16.

Table 16. IBM-supplied Cataloged Procedures

Procedure	Name	For sample coding, see
Link-edit and run an application written in any HLL supported by LE/370	CEEWLG	"CEEWLG — Link and Run a Program Written in an LE/370-conforming HLL" on page 64
Link-edit an application written in any HLL supported by LE/370	CEEWL	"CEEWL — Link an Application Written in an LE/370-conforming HLL" on page 65
Load and run an application written in any HLL supported by LE/370	CEEWG	"CEEWG — Load and Run a Program Written in an LE/370-conforming HLL" on page 65
Compile and link-edit a C application	EDCCL	*IBM SAA AD/Cycle C/370 Programming Guide*
Compile, link-edit, and run a C application	EDCCLG	*IBM SAA AD/Cycle C/370 Programming Guide*
Compile, prelink, link-edit, and run a C application	EDCCPLG	*IBM SAA AD/Cycle C/370 Programming Guide*
Prelink and link-edit a C application	EDCPL	"EDCPL — Prelink and Link a C/370 Application" on page 66
Invoke the C/370 Object Library Facility	EDCLIB	"EDCLIB — Invoke the Object Library Facility" on page 68
Compile and link-edit a COBOL application	IGYWCL	*IBM SAA AD/Cycle COBOL/370 Programming Guide*
Compile, link-edit, and run a COBOL application	IGYWCLG	*IBM SAA AD/Cycle COBOL/370 Programming Guide*
Compile, load, and run a COBOL application	IGYWCG	*IBM SAA AD/Cycle COBOL/370 Programming Guide*

This chapter describes two cataloged procedures for link-editing and running, and loading and running your application that can be used with any LE/370-conforming HLL. Two C/370-specific catalogued procedures are also described. All other HLL-specific cataloged procedures that contain a compile step are described in the appropriate HLL Programming Guide listed in "Bibliography" on page 477.

Step Names in Procedures

The *stepname* in a cataloged procedure is the same as the abbreviated processor name. For example, the step that executes a compiled and link-edited application is named GO. In the procedure named CEEWLG (see "CEEWLG — Link and Run a Program Written in an LE/370-conforming HLL" on page 64), the first step is named LKED, and the second is named GO.

Unit Names in Procedures

The generic unit name used in IBM-supplied cataloged procedures is:

```
data set  DD  UNIT=SYSDA      Any direct-access device
```

A pool of units must be assigned to SYSDA during the system generation procedure. After a pool of devices is assigned to this class, device selection is done by the job scheduler.

Data Set Names in Procedures

When DSNAME=&&*name* is used in a DD statement, it is assumed to be a temporary data set that will be deleted when the job terminates. If the data set is to be kept, override the DD statement with a permanent data set name and specify the appropriate DISP parameters.

See Chapter 9, "Using the Linkage-Editor Under MVS" on page 31 for a detailed description of each of the data sets included in the cataloged procedures below. See "Overriding and Adding to EXEC Statements" on page 70 for instructions about overriding DD statements in cataloged procedures.

CEEWLG — Link and Run a Program Written in an LE/370-conforming HLL

The CEEWLG cataloged procedure shown in Figure 26 includes two procedure steps:

- LKED — invokes the linkage editor (symbolic name HEWL) to link-edit an object module.
- G0 — executes the load module produced in the first step.

```
//CEEWLG   PROC LIBPRFX='CEE.V1R1M0',GOPGM=GO
//LKED     EXEC PGM=HEWL,REGION=1024K
//SYSLIB    DD  DSNAME=&LIBPRFX..SCEELKED,DISP=SHR
//SYSPRINT  DD  SYSOUT=*
//SYSLIN    DD  DDNAME=SYSIN
//SYSLMOD   DD  DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
//              UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1    DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED),REGION=2048K
//STEPLIB   DD  DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
//SYSPRINT  DD  SYSOUT=*
//CEEDUMP   DD  SYSOUT=*
//SYSABOUT  DD  SYSOUT=*
//SYSDBOUT  DD  SYSOUT=*
//SYSUDUMP  DD  SYSOUT=*
```

Figure 26. Cataloged Procedure CEEWLG. Link-edit and Run a Program Written in Any LE/370-conforming HLL

Notes

1. The following DD statement, indicating the location of the object module, must be supplied in the input stream:

   ```
   //LKED.SYSIN  DD  *         (or appropriate parameters)
   ```

2. The data set SCEELKED must be included in your link-edit SYSLIB concatenation. This is the name of the LE/370 link-edit library. (The high-level qualifier of this link-edit library may be changed at your installation).

The data set SCEERUN must be included in the STEPLIB DD statement for the GO step. (The name of this load library may be changed at your installation).

3. If the application refers to any data sets in the execution step (such as user-defined files or SYSIN), DD statements that define these data sets must also be provided.

CEEWL — Link an Application Written in an LE/370-conforming HLL

The CEEWL cataloged procedure shown in Figure 27 includes one procedure step:

- LKED — invokes the linkage editor (symbolic name HEWL) to link-edit an object module.

```
//LKED    EXEC PGM=HEWL,REGION=1024K
//SYSLIB    DD  DSNAME=&LIBPRFX..SCEELKED,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN    DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&PGMLIB(&GOPGM),
//              SPACE=(TRK,(10,10,1)),
//              UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1    DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
```

Figure 27. Cataloged Procedure CEEWL. Link-edit an Application Written in Any LE/370-conforming HLL

Notes

1. The following DD statement, indicating the location of the object module, must be supplied in the input stream:

   `//LKED.SYSIN  DD  *        (or appropriate parameters)`

2. The data set SCEELKED must be included in your link-edit SYSLIB concatenation. This is the name of the LE/370 link-edit library. (The high-level qualifier of this link-edit library may be changed at your installation).

CEEWG — Load and Run a Program Written in an LE/370-conforming HLL

The CEEWG cataloged procedure shown in Figure 28 includes the following step:

- GO — loads an object module produced by the compiler and executes the load module.

```
//CEEWG     PROC LIBPRFX='CEE.V1R1M0'
//GO      EXEC PGM=LOADER,REGION=2048K
//SYSLIB    DD  DSNAME=&LIBPRFX..SCEELKED,DISP=SHR
//SYSLOUT   DD  SYSOUT=*
//SYSLIN    DD  DDNAME=SYSIN
//STEPLIB   DD  DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP   DD  SYSOUT=*
//SYSABOUT DD  SYSOUT=*
//SYSDBOUT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

Figure 28. Cataloged Procedure CEEWG. Load and Run a Program Written in Any LE/370-conforming HLL

Notes

1. The following DD statement, indicating the location of the object module, must be supplied in the input stream:

```
//GO.SYSIN  DD   *        (cr appropriate parameters)
```

2. The data set SCEELKED must be included in your link-edit SYSLIB concatenation. This is the name of the LE/370 resident library. (The high level qualifier of this resident library may be changed at your installation).

   The data set SCEERUN must be included in the STEPLIB DD statement for the GO step. (The name of this load library may be changed at your installation).

3. If the application refers to any data sets in the execution step (such as user-defined files or SYSIN), DD statements that define these data sets must also be provided.

EDCPL — Prelink and Link a C/370 Application

The EDCPL cataloged procedure shown in Figure 29 includes two procedure steps:

- PLKED — invokes the pre-link facility linkage editor (symbolic name EDCPRLK) to pre-link an object module.
- LKED — invokes the linkage editor (symbolic name HEWL) to link-edit an object module.

```
//*********************************************************************
//*                                                                   *
//* LICENSED MATERIALS - PROPERTY OF IBM                              *
//*                                                                   *
//* 5688-198 (C) COPYRIGHT IBM CORP. 1991                             *
//* ALL RIGHTS RESERVED                                               *
//*                                                                   *
//* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                      *
//* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP                   *
//* SCHEDULE CONTRACT WITH IBM CORP                                   *
//*                                                                   *
//* SEE COPYRIGHT INSTRUCTIONS                                        *
//*                                                                   *
//*********************************************************************
//*                                                                   *
//* IBM SAA AD/CYCLE LANGUAGE ENVIRONMENT/370                         *
//*                                                                   *
//*  PRE-LINK (REQUIRED FOR REENTRANCY) AND LINK EDIT                 *
//*  A C PROGRAM                                                      *
//*                                                                   *
//*  RELEASE LEVEL: 01.01.00  (VERSION.RELEASE.MODIFICATION LEVEL)    *
//*                                                                   *
//*  PARAMETER  DEFAULT VALUE    USAGE                                *
//*   INFILE    NONE             INPUT DATA SET                       *
//*   OUTFILE   &&GSET(GO)       OUTPUT LINK EDIT DATA SET            *
//*   SYSLBLK   3200             BLOCKSIZE FOR &OUTFILE               *
//*   PREGSIZ   2048K            PRE-LINKER REGION SIZE               *
//*   PPARM     NONE             PRE-LINKER OPTIONS                   *
//*   LIBPRFX   CEE.V1R1M0       PREFIX FOR LIBRARY DATA SET NAMES    *
//*   LREGSIZ   1024K            LINK EDIT REGION SIZE                *
//*   LPARM     AMODE=31,MAP     LINK EDIT OPTIONS                    *
//*                                                                   *
//*********************************************************************
//*
//EDCPL  PROC INFILE=,
// OUTFILE='&&GSET(GO),DISP=(MOD,PASS),UNIT=VIO,SPACE=(512,(50,20,1))',
//   SYSLBLK='3200',
//   PREGSIZ='2048K',
//   PPARM=,
//   LIBPRFX='CEE.V1R1M0',
//   LREGSIZ='1024K',
//   LPARM='AMODE=31,MAP'
//*
//*--------------------------------------------------------------
//* PRE-LINKEDIT STEP:
//*--------------------------------------------------------------
//PLKED    EXEC PGM=EDCPRLK,PARM='&PPARM',
//    REGION=&PREGSIZ
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB   DD  DUMMY
//SYSIN    DD  DSN=&INFILE,DISP=SHR
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=VIO,DISP=(MOD,PASS),
//             SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=&SYSLBLK)
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//*--------------------------------------------------------------------
//* LINKEDIT STEP:
//*--------------------------------------------------------------------
//LKED   EXEC PGM=HEWL,COND=(8,LE,PLKED),
//    REGION=&LREGSIZ,PARM='&LPARM'
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=*.PLKED.SYSMOD,DISP=SHR
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&OUTFILE
//SYSUT1   DD  UNIT=VIO,SPACE=(32000,(30,30))
```

Figure 29. Cataloged Procedure EDCPL. Pre-link and Link a C/370 Application

Notes

- The following DD statement, indicating the location of the object module, must be supplied in the input stream:

```
//LKED.SYSIN  DD   *        (or appropriate parameters)
```

- The data set SCEELKED must be included in your link-edit SYSLIB concatenation. This is the name of the LE/370 link-edit library. (Note, however, that the high-level qualifier of this link-edit library may be changed at your installation).
- Specify your input C/370 source file in the INFILE parameter. If you do not specify the input data set name, you must use JCL statements to override the appropriate SYSLIN DD statement of the cataloged procedure.
- Specify the output C/370 module name (that is, the file where the load module is to be stored) and file characteristics in the OUTFILE parameter.

  If you do not specify an OUTFILE name, a temporary data set will be generated to hold the load module.
- If two contradictory pre-link utility options are specified in the)PARM parameter, the last is accepted and the first is ignored. For a list of pre-link options see Table 57 on page 438.

For more information see Appendix A, "Pre-linking Your C Application" on page 431.

EDCLIB — Invoke the Object Library Facility

The EDCLIB cataloged procedure shown in Figure 30 includes one procedure step:

- OUTILITY — Invokes the C/370 Library Facility.

```
//*********************************************************************
//*                                                                   *
//* LICENSED MATERIALS - PROPERTY OF IBM                              *
//*                                                                   *
//* 5688-198 (C) COPYRIGHT IBM CORP. 1991                             *
//* ALL RIGHTS RESERVED                                               *
//*                                                                   *
//* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                      *
//* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP                   *
//* SCHEDULE CONTRACT WITH IBM CORP                                   *
//*                                                                   *
//* SEE COPYRIGHT INSTRUCTIONS                                        *
//*                                                                   *
//*********************************************************************
//*                                                                   *
//* IBM SAA AD/CYCLE LANGUAGE ENVIRONMENT/370                         *
//*                                                                   *
//*  RUN THE OBJECT LIBRARY UTILITY                                   *
//*                                                                   *
//*  RELEASE LEVEL: 01.01.00  (VERSION.RELEASE.MODIFICATION LEVEL)    *
//*                                                                   *
//*  PARAMETER  DEFAULT VALUE    USAGE                                *
//*   OREGSIZ   2048K            OBJECT LIBRARY REGION SIZE           *
//*   LIBRARY   NONE             OBJECT LIBRARY PDS                   *
//*   OPARM     NONE             OBJECT LIBRARY UTILITY PARAMETERS    *
//*   OBJECT    DUMMY            ADD FUNTION ONLY: INPUT OBJ MODULE   *
//*   LIBPARM   PO,80            PARAMETERS FOR DD FOR OUTPUT PDS     *
//*   LIBPRFX   CEE.V1R1M0       PREFIX FOR LIBRARY DATA SET NAMES    *
//*                                                                   *
//*********************************************************************
//*
//EDCLIB  PROC OREGSIZ='2048K',
//   LIBRARY=,
//   OPARM=,
//   OBJECT=DUMMY,
//   LIBPARM='DCB=(DSORG=PO,LRECL=80)',
//   LIBPRFX='CEE.V1R1M0'
//*
//*-------------------------------------------------------------
//* LIBRARY UTILITY STEP:
//*-------------------------------------------------------------
//OUTILITY  EXEC PGM=EDCALIAS,PARM='&OPARM',
//    REGION=&OREGSIZ
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
//SYSIN    DD  &OBJECT
//SYSLIB   DD  DSNAME=&LIBRARY,&LIBPARM,DISP=SHR
//SYSOUT   DD  SYSOUT=*
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSPRINT DD  SYSOUT=*
```

Figure 30. Cataloged Procedure EDCLIB. Invoke the C/370 Library Facility

Notes

- Use this procedure to generate an object library TXTLIB, to add or delete members, or to list members in the object library.
- The OBJECT parameter contains the object module to be added to the library. The data set name (DSN=) and any applicable keyword parameters (such as DCB, DISP) can be specified using this parameter.

  The default is OBJECT=DUMMY.

  OBJECT=DUMMY is required if you plan to add a module to the library.
- The LIBRARY parameter contains the data set name for the library for a requested function (ADD, DEL, MAP, or DIR). An example is LIBRARY='PROGRAM1.LIB.OBJ'. The MEMBER parameter contains the member of the library to contain the object module. An example is MEMBER='MYPROG'.

- The LIBPARM parameter contains keyword parameters for the library. The default is LIBPARM='DCB=(DSORG=PO,LRECL=80)'.
- If two contradictory object library utility parameters are specified in the OPARM parameter, the last is accepted and the first is ignored.

For more information, see Appendix C, "Using the C/370 Object Library Utility" on page 453.

Modifying Cataloged Procedures

You can modify the statements of a cataloged procedure for the duration of a job step in which it is invoked, either by adding DD statements to the procedure or overriding one or more parameters in the EXEC or DD statements. Any parameter in a cataloged procedure, except the PGM=*progname* parameter in the EXEC statement, can be overridden. Parameters or statements not specified in the procedure can also be added. When a cataloged procedure is overridden or added to, the changes apply only during that one execution. The changes do not affect the master copy of the cataloged procedure stored in the procedure library.

Overriding and Adding to EXEC Statements

A parameter with a qualified name (that is, qualified by the procedure step in which it is specified) applies only to the EXEC statement in which it was specified. If a parameter of an EXEC statement that invokes a cataloged procedure has an unqualified name, the parameter applies to all the EXEC statements in the cataloged procedure. For example, `REGION=2048` specifies a region size of 2048 for all of the EXEC statements in a given procedure, whereas `REGION.GO=2048` applies to only the GO step of the procedure.

If you want to modify a multistep procedure, you can do so by specifying parameters with qualified names on a step by step basis. If you want to modify the entire procedure, specify the name of the parameter in an EXEC statement without qualifying it. The modifications override existing parameters in the cataloged procedure.

Overriding and Adding DD Statements

A DD statement can be overridden or added by specifying a DD statement whose name is composed of the *ddname* of the DD statement being overridden, preceded by the procedure *stepname* that qualifies that *ddname*:

//procstep.ddname DD (*appropriate parameters*)

Two rules must be followed when overriding or adding a DD statement within a step in a procedure.

1. Overriding DD statements must be in the same order in the input stream as they are in the cataloged procedure.
2. DD statements to be added must follow overriding DD statements.

There are some special cases that you should keep in mind when overriding a DD statement.

- To nullify a keyword parameter (except the DCB and AMP parameters), write, in the overriding DD statement, the keyword and an equal sign followed by a

comma. For example, to nullify the use of the UNIT parameter, specify UNIT=, in the overriding DD statement.

- A parameter can be nullified by specifying a mutually exclusive parameter. For example, the SPACE parameter can be nullified by specifying the SPLIT parameter in the overriding DD statement.
- To override DD statements in a concatenation of data sets, you must provide one DD statement for each data set in the concatenation. Only the first DD statement in the concatenation should be named. However, if a DD statement to be changed follows one (or more) DD statement(s) to be left intact, the first overriding statement(s) should have a blank operand.
- If the DDNAME=*ddname* parameter is specified in a cataloged procedure, it cannot be overridden; rather, it can refer to a DD statement supplied at the time of execution.

The following example:

- Specifies PROGRAM1 in USER.OBJLIB an the input object module to the link-editor
- Changes the library prefix for the SCEELKED link library to SYS1
- Increases the region for linking and running the application
- Passes the RPTSTG and RPTOPTS options to the load module when it is executed in the GO step of the procedure.

```
//CEEWLG   JOB
//*
//LINKGO EXEC CEEWLG,
//  LIBPRFX='SYS1',
//  REGION=2048K,
//  PARM.GO='RPTSTG(ON) RPTOPTS(ON)/'
//*
//LKED.SYSIN   DD DSN=USER.OBJLIB(PROGRAM1),DISP=SHR
//*
```

Figure 31. Overriding Parameters in the CEEWLG Cataloged Procedure

For a discussion of considerations for running ILC applications under LE/370, refer to Chapter 25, "C/370—COBOL Interlanguage Communication" on page 140.

Managing the Language Environment

LE/370 performs some commonly used functions which affect how your program executes in the common run-time environment. Note that a number of the callable services documented in this guide permit you to tailor features of the environment such as storage management or condition handling to suit your application. This section describes general information about LE/370 which will help you to decide how to modify the environment in which your application will run.

The following topics are discussed:

Chapter 17. Stack and Heap Storage 74
Stack Storage 74
Heap Storage 76

Chapter 18. Communicating Conditions 79
Condition Token Layout 79
Testing a Condition Token 80
Relationship of the Condition Token to Messages 80
Messages and the Callable Service Feedback Code 81

Chapter 19. Condition Management 83
Sources of Conditions 83
Obtaining the Stack Frame 83
Handlers Invoked During Condition Handling 84
Responses to Conditions 85
Condition Management Model 86
Condition Handling Example 88
ERRCOUNT Run-time Option 90
Language-specific Condition Handling Semantics 90
C/370 Condition Handling Semantics 91
COBOL Condition Handling Semantics 94
ILC Condition Handling Semantics 95
Nested Conditions 97
LE/370-issued ABENDs 97
Using XUFLOW and CEE3SPM to Enable and Disable Hardware Conditions . 98
User-Written Condition Handler 98
Callable Services Available to Handle Conditions 100
Examples 100

Chapter 20. Message Handling 110
Delivering the Message 110
Dynamic Allocation of the MSGFILE ddname 110
Inserting Messages in Your Application 111
Other Message Handling Run-time Options and Callable Services 114
Message Handling and National Language Support 115
Specifying the National Language 115
Specifying the National Country Setting 115
Using COBOL/370 DISPLAY and ACCEPT statements 116
DISPLAY statement 116
ACCEPT Statement 117

© Copyright IBM Corp. 1991

Chapter 17. Stack and Heap Storage

The LE/370 storage manager provides services that control the stack and heap storage used at run time. LE/370-conforming HLLs and assembler language routines use these services for all storage requests.

Stack Storage

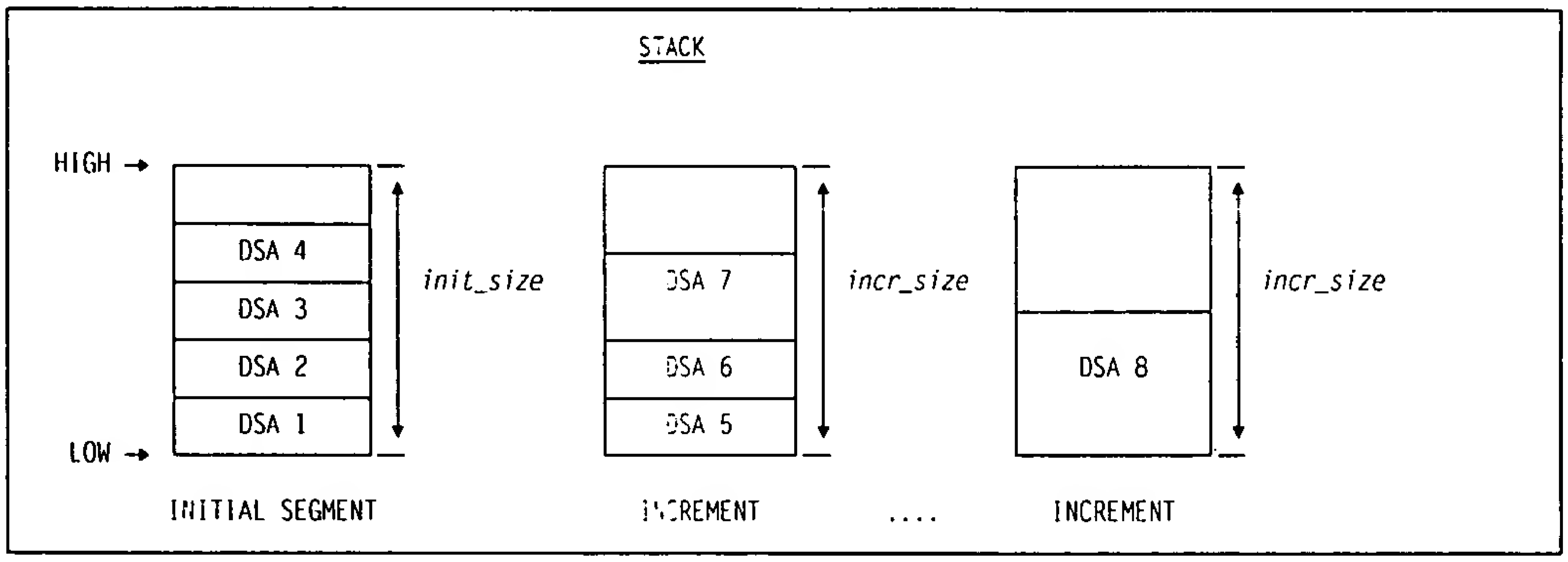


Figure 32. LE/370 Stack Storage Model.

Refer to Figure 32 in the following discussion of the LE/370 stack storage model.

Stack storage is comprised of an *initial stack segment* that is allocated during enclave initialization, and *stack increments* that are allocated whenever the initial stack segment becomes full. The size of the initial stack segment and the increment are specified by the *init_size* and *incr_size* parameters of the STACK run-time option. If the STACK option is not specified, LE/370 uses the installation default or application default as the initial stack segment size (see Chapter 30, "Specifying Run-time Options" on page 206 for more information).

The STACK contains the *DSA*s associated with each thread in the common run-time environment. A DSA is storage acquired during the execution of the program and is comprised of a standard register save area and an area available for dynamic storage allocation. A DSA is allocated every time a procedure, function, or block is entered, as,for example, when a call is made to an LE/370 service routine. A DSA is freed when the procedure or block returns control. As DSAs are allocated, the initial stack segment fills. When the initial stack segment becomes full, overflow stack segments or *increments* are obtained. The size of the "increments" is governed by the value of the *incr_size* parameter of the STACK run-time option.

Stack storage is managed on two stacks, the *user stack* and the *library stack*. The primary stack is the user stack. It manages storage residing above or below the 16M line, depending on the setting of the STACK run-time option. The secondary library stack is always directed below the 16M line. It manages storage used by library routines needing DSAs below the 16M line, and is controlled by the LIBSTACK run-time option. LE/370 routines may acquire library DSAs from the user stack, but user code never acquires DSAs from the library stack.

3 Except under CICS when the ALL31(ON) option is in effect.

© Copyright IBM Corp. 1991

All stack storage is managed at the thread level; each thread "owns" its stack. Therefore, stack storage is not shared between threads.

Stack Terminology

Stack Storage
: Program storage allocated on a LIFO basis. The stack consists of one or more stack segments.

Segment
: A large, contiguous area of storage obtained directly from the operating system. The LE/370 storage management scheme subdivides these large stack segments into numerous individual DSAs. The first segment used for stack storage is called the initial stack segment. If the initial stack segment becomes full, a second segment or stack increment is obtained from the operating system.

Increment
: The second and subsequent segments of storage allocated to the stack, when storage in the initial stack segment is used up.

Dynamic Storage Area (DSA or Stack Frame)
: Dynamically acquired storage composed of a register save area and an area available for dynamic storage allocation (that is, program variables). DSAs are added to the stack when a routine is entered, and removed upon exit (last-in, first-out).

Tuning the Stacks

For best performance, the initial stack segment should be large enough to satisfy all requests for stack storage. On the other hand, an initial stack segment that is too large may waste storage. This may degrade overall system performance, especially under CICS where storage is limited. The LE/370 storage report generated by the RPTSTG(ON) option ("RPTSTG" on page 240) shows you how much stack storage is being used, the total number of segments allocated to the stack (GETMAINs), and the recommended values for the STACK run-time option.

Note: RPTSTG(ON), as well as the STORAGE run-time option (see "STORAGE" on page 246) can have a negative impact upon the performance of your application. Therefore, always specify the IBM-supplied default setting RPTSTG(OFF) when running production jobs. Use RPTSTG(ON) and STORAGE(NONE,NONE,NONE) only to debug and/or tune applications.

Heap Storage

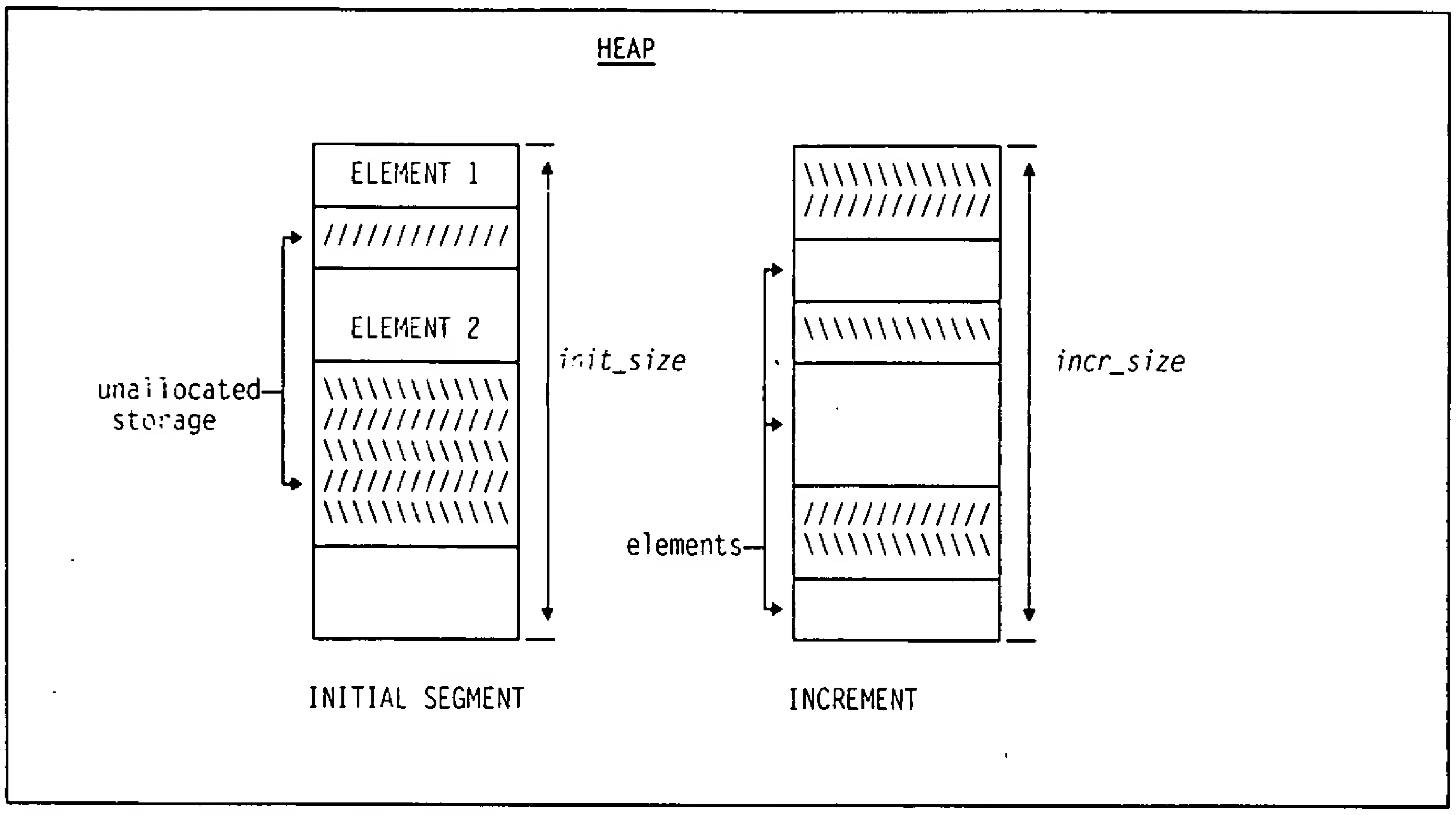


Figure 33. LE/370 Heap Storage model

Refer to Figure 33 in the following discussion of the LE/370 heap storage model.

The LE/370 heap storage model allows multiple heaps that are dynamically created and discarded. Unlike stack storage, a heap is an area of storage used for suballocations of dynamic storage whose lifetime is not related to the execution of a particular routine. Heap storage is composed of an *initial heap segment* that is allocated by the first request for heap storage, and a *heap increment* that is allocated when any subsequent increment to the initial heap segment is required. An increment is required whenever insufficient free space remains in the heap to satisfy the current request for storage. The size of the initial heap segment is governed by the *init_size* parameter of the HEAP run-time option (see "HEAP" on page 230). The *incr_size* parameter governs the size of the heap increment.

Heap storage is shared among all program units and all threads in an enclave. Allocated heap storage remains allocated until it is explicitly freed or until the enclave terminates. Any thread may free the heap storage. This may be done one element at a time with the CEEFRST callable service, or all heap elements may be freed at once using CEEDSHP. The *initial* heap cannot be discarded, however.

Additional heaps provide isolation between logical groups of data in different additional heaps. Use separate additional heaps whenever you foresee the need to group storage objects together so that they can be freed at once (with a single call to CEEDSHP), rather than one element at a time (with calls to CEEFRST).

Allocation and freeing of heap storage is typically controlled by the programmer. The HEAP run-time option allows you to control the size of the initial allocation and any subsequent increments to the heap In addition, LE/370 provides callable services for all heap operations. These services fall into two broad categories:

- Basic heap operations:
  - ***CEEGTST***— allocate heap storage element (**G**e**T** **ST**orage).
  - ***CEEFRST***— free heap storage element (**FR**ee **ST**orage).
  - ***CEECZST***— reallocate heap storage element (**C**hange si**Z**e of **ST**orage).

- Extended heap operations (multiple heap support).
  - ***CEECRHP***— create a new heap (**CR**eate **HeaP**).
  - ***CEEDSHP***— discard an entire heap (**DiS**card **HeaP**).

See Chapter 34, "Dynamic Storage Callable Services" on page 267 for further information about the above callable services.

In addition to the HEAP run-time option, LE/370 offers the following options to control allocation of library heap storage:

- ***ANYHEAP***— control allocation of library (HLL and LE/370) heap storage not restricted to below the 16M line.
- ***BELOWHEAP***— control allocation of library heap storage below the 16M line.

Note: The LE/370 Anywhere Heap and Below Heap are reserved for run-time library usage only. Application data and variables are not kept in these heaps. You normally should not adjust the size of these heaps unless the storage report indicates excessive GETMAINS for the Anywhere or Below Heaps.

Table 17 lists LE/370 heaps and their respective purposes.

Table 17. Heap IDs Recognized by LE/370 Heap Manager

Heap Name	Heap-id	Intended Purpose	Created by	Disposed by
Initial heap, or User Heap	0	Application program data. Common heap used by C language intrinsic functions and COBOL variables. COBOL access is by LE/370 callable services. CEEDSHP has no effect on the initial heap.	Enclave Initialization. Size and location determined from HEAP run-time option.	Enclave Termination
Additional Heaps	(returned by **CEECRHP**)	Collections of application program data that may be quickly disposed with a single **CEEDSHP** call.	Call **CEECRHP**. Arguments define heap size and location.	Call **CEEDSHP**, or Enclave Termination
LE/370 Anywhere Heap	n/a	Used internally by LE/370 routines. Not used for user application data.	Enclave Initialization	Enclave Termination
LE/370 Below Heap	n/a	Used internally by LE/370 routines Not used for user application data.	Enclave Initialization	Enclave Termination

AMODE Considerations

The *initsz24* and *incrsz24* parameters of the HEAP run-time option control the initial size and subsequent increments of heap storage allocated below the 16M line. This storage is required for AMODE(24) applications running with the ALL31(OFF) and HEAP(,,ANYWHERE) run-time options in effect.

For example, suppose the initial heap segment is allocated above the 16M line. If an AMODE(24) routine requests storage from this initial heap, LE/370 must allocate a heap segment from below the 16M line so that the AMODE(24) routine can address the storage. The size of this heap segment and any subsequent segments that are needed is determined by the values specified in the *initsz24* and *incrsz24* parameters of HEAP.

Tuning the Heap

For best performance, the initial heap segment should be large enough to satisfy all requests for heap storage. The LE/370 storage report generated by the RPTSTG(ON) run-time option ("RPTSTG" on page 240) shows you how much heap storage is being used, the total number of segments allocated to the heap (GETMAINs), and the recommended values for the HEAP, ANYHEAP, and BELOWHEAP run-time options.

Note: RPTSTG(ON), as well as the STORAGE run-time option (see "STORAGE" on page 246) can have a negative impact on the performance of your application. Therefore, always specify the IBM-supplied default setting RPTSTG(OFF) when running production jobs. Use RPTSTG(ON) and STORAGE(NONE,NONE,NONE) only to debug applications.

Heap Terminology

Heap storage
A collection of one or more heap segments consisting of the initial heap segment and zero or more increments.

Initial heap segment
A contiguous area of storage obtained directly from the operating system. Heap elements are allocated within each segment by the LE/370 storage management routines. The LE/370 storage management scheme subdivides heap segments into individual heap elements. Also termed the *user heap* or *heap 0*.

Heap increment
Additional heap segments allocated when the initial heap segment becomes full.

Heap element
A contiguous area of storage allocated by a call to the CEEGTST service. Heap elements are always allocated within a single heap segment. An element usually contains a single program variable, array, or data structure.

Chapter 18. Communicating Conditions

LE/370 uses a 12-byte *condition token* data type to perform a variety of communication functions. The condition token communicates with message services, callable services, and the condition manager. The information returned in the optional *fc* parameter of all LE/370 callable services, for example, is communicated using a condition token.

Condition Token Layout

```
                                                                                      Bits
                                                                                        |
                                                                                        v
                                  3 33 3 3 3 3 4                  6                     9
0                                 1 23 4 6 7 9 0                  3                     5
+---------------------------------+--+--+---+--------------------+---------------------+
|      Condition_ID               |A |B  | C |     Facility_ID    |     I_S_Info        |
+---------------------------------+--+--+---+--------------------+---------------------+

    A = Case
    B = Severity
    C = Control

Cases of Condition_ID are:

Case 1
                   1 1          3
    0              5 6          1
   +----------------+----------------+
   | Severity       |     Msg_No     |
   +----------------+----------------+

Case 2
                   1 1          3
    0              5 6          1
   +----------------+----------------+
   |   Class_Code   |   Cause_Code   |
   +----------------+----------------+
```

Figure 34. LE/370 condition token

Figure 34 contains a mapping of the condition token. Every condition token contains the components indicated in Figure 34:

Condition_ID A 4-byte identifier that, with the Facility_ID, describes the condition that the token will communicate. The format of the Condition_ID field depends on whether a Case 1 (service condition) or Case 2 (Class/Code) condition is being represented. LE/370 callable services and most applications can produce Case 1 conditions. Case 2 conditions are produced by some operating systems and compiler libraries.

Figure 34 illustrates the format of the Condition_ID for Case 1 and Case 2 conditions.

Case Specifies whether the condition token is for a Case 1 or Case 2 condition.

Severity Specifies the severity of the condition represented by the condition token.

Msg_No Identifies the message associated with the condition.

© Copyright IBM Corp. 1991

Class_Code Identifies the message id associated with the class of the condition.

Cause_Code Identifies the message id associated with the cause of the condition.

Control Specifies whether the Facility_ID has been assigned by IBM.

Facility_ID Identifies the facility that generated the condition; in the case of LE/370, the Facility_ID is CEE. Although all LE/370-participating HLLs use LE/370 message and condition handling services, the actual run-time messages generated under LE/370 still carry the language identification in the Facility_ID. The Facility_ID for COBOL, for example, is IGZ.

I_S_Info Identifies the Instance Specific Information associated with a given instance of the condition.

For a detailed explanation of each of the fields in the condition token and information about how to construct and deconstruct your own condition token, see "CEENCOD — Construct a Condition Token" on page 306 and "CEEDCOD — Decompose a Condition Token" on page 287, respectively.

Testing a Condition Token

You can test a condition token that is returned from a callable service for the following:

Success To test for success, determine if the first 4 bytes are zero. If the first 4 bytes are zero, the remainder of the condition token is zero, indicating a successful call was made to the service.

Equivalent Tokens To determine whether two condition tokens are equivalent (that is, the same "type" of condition token, but not the same "instance" or occurrence of the condition token), compare the first 8 bytes of each condition token with one another. These bytes are static and do not change depending upon the given instance when the condition occurs.

Equal Tokens To determine whether two condition tokens are equal, (that is, the same "instance" or occurrence of the condition token), compare all 12 bytes of each condition token with one another. The last 4 bytes can change from instance to instance of a condition token.

Relationship of the Condition Token to Messages

For every condition that is raised in LE/370, there is also a message associated with that condition, as represented by the condition token. The condition token contains a unique id that LE/370 uses to write a message associated with the condition to the MSGFILE. Every run-time message in LE/370 is comprised of:

- A 3-character Facility ID used by all messages generated under LE/370 or a particular LE/370-conforming product. For example, all messages issued by LE/370 have a Facility ID of "CEE".
- A message number that identifies the message associated with the condition.

- A severity level that indicates to the operating system the degree of success or failure of the current execution.

The format of every run-time message is as follows:

```
FFFxxxxM
```

where:

FFF
: is the Facility_ID. In Language Environment/370 Version 1, the possible Facility_IDs are:

- CEE — LE/370 common library
- IGZ — COBOL language-specific library
- EDC — C language-specific library

xxxx
: Error message number

M Message Class

The message class is represented by a single character that indicates the severity class of the message. The severity of the condition represented by the condition token is specified according to the values listed in Table 18.

Table 18. LE/370 Message Classes and Associated Severity Codes

Message Class	Severity Code	Explanation
I	0	An informational message (or, if the entire token is zero, no information).
W	1	A warning message. Service completed, probably correctly.
E	2	An error message - correction attempted. Service completed, perhaps incorrectly.
S	3	A severe error message. Service not completed.
C	4	A critical error message. Service not completed. Condition signaled. A critical error is a condition that jeopardizes the environment. If a critical error occurs during an LE/370 callable service, it is always signaled to the condition manager instead of being returned synchronously to the caller.

A list of run-time messages generated under LE/370 is presented in the *Language Environment/370 Debugging and Run-time Messages Guide.*

Messages and the Callable Service Feedback Code

If you code the *fc* parameter in your application in order to receive feedback information from a callable service and a condition is raised, the following sequence of events occurs:

1. The callable service in which the condition occurred builds a condition token for the condition.

2. The callable service places information into the Instance Specific Information (ISI) area. The ISI contains the following:

   - Information that will be inserted into a message associated with the condition
   - Information that will be used by the LE/370 condition handler to identify and react to the condition.

3. If the severity of the detected condition is critical (severity = 4), it is signaled directly to the condition manager. The condition manager then processes the condition in the manner described in Chapter 19, "Condition Management" on page 83.

4. If the condition severity is not critical (severity less than 4), the condition token is returned to the routine that called the service.

5. When the condition token is returned to your application, you have the following options:

   - Ignore it and continue processing
   - Signal it to the condition manager using the CEESGL callable service (see "CEESGL — Signal a Condition" on page 311)
   - Get, format, and dispatch the message for display using the CEEMSG callable service (see "CEEMSG — Get, Format, and Dispatch a Message" on page 331)
   - Store the message in a storage area using the CEEMGET callable service (see "CEEMGET — Get a Message" on page 327)
   - Use the CEEMOUT callable service (see "CEEMOUT — Dispatch a Message" on page 330) to dispatch a user-defined message to a destination that you specify.

If you omit the *fc* parameter when you are calling a service, the callable service signals the condition manager immediately upon detection of the condition.

Note: In C/370, you omit a parameter by passing a null pointer to the callable service in place of the parameter. In COBOL, however, you *must* specify the *fc* parameter when you call an LE/370 service. The *fc* parameter is required when an LE/370 callable service is called by a COBOL CALL statement.

Chapter 19. Condition Management

The LE/370 condition manager facilitates the handling of conditions in the run-time environment. The main purpose of a condition manager is to diagnose a problem, then to determine how to deal with it. LE/370-conforming HLLs, applications, and debug tools each have their own condition handling routines that are invoked to determine the correct action for handling errors. LE/370 condition management does not replace current HLL condition handling, but provides a uniform way to handle conditions which occur in the run-time environment for applications written in all LE/370-conforming languages.

It is important to become familiar with LE/370 condition handling so that you understand how your application reacts when conditions arise. The following sections describe condition handling under LE/370.

The "Examples" on page 100 contain two sample applications which illustrate features of the LE/370 condition handling model.

Sources of Conditions

A condition is any synchronous event that can require the attention of an executing application or the language routines supporting it. Conditions can originate from any of the following:

- **Hardware detected errors** (also known as program interruptions) are signaled by the central processing unit. The error codes are derived from those defined for the machine on which the application is executing.
- **Operating system detected errors** (also known as exceptions) are software errors and are reported to you as ABENDs on MVS and VM.
- **Software generated signals** are conditions intentionally and explicitly created by LE/370, language library routines, applications, or condition handling routines. The CEESGL callable service, for example, is used to create signals (see "CEESGL — Signal a Condition" on page 311 for more information). Software generated signals are handled by the LE/370 condition manager in the same manner as errors detected by the hardware or operating system.

Obtaining the Stack Frame

The LE/370 condition management model is a *stack frame*-based model. A stack frame includes an 18 fullword register save area in addition to other information and is backchained to a previous stack frame.

Stack frames are obtained from stack storage every time a routine or procedure is entered, for example, when a call is made to a service routine. Stack storage is obtained and kept in a subpool. The size of the stack storage is controlled by the STACK run-time option ("STACK" on page 245). Each time a stack frame is needed, it is obtained from this subpool. If not enough stack storage is available for a request, then an explicit storage allocation is performed.

Each new obtained stack frame is chained in LIFO order on the stack. Condition handling mechanisms such as HLL and user-written condition handlers can be

© Copyright IBM Corp. 1991

associated with the stack frames. LE/370 can therefore use the stack to keep track of the various HLL and user-written condition handlers that can potentially be invoked to handle conditions for a given application.

Stack frames are obtained differently in different languages. The following causes a stack frame to be obtained:

- A function call in C
- Entry into a block in C
- Entry into a compile-unit in COBOL.

Note that stack frames may be associated with library routines in addition to routines within user applications.

Not all LE/370-conforming HLLs obtain stack frames off the stack. COBOL/370, for example uses a control block contained in the application's object module or dynamically allocated from heap storage. Stack frames are also commonly known as:

- Activation Records
- Save Areas
- DSAs (Dynamic Save Areas)
- DSAs (Dynamic Storage Areas)
- Register Save Areas.

The term "stack frame" will be used throughout this chapter as a rough equivalent to the above terms.

Under LE/370, stack frame 0 (zero) is a "conceptual" stack frame where the thread is initialized and the first routine is called.

Handlers Invoked During Condition Handling

There is a distinction between the condition manager and condition handlers. LE/370 provides a condition *manager* that manages conditions in the common execution environment by invoking various condition *handlers*. Some of these condition handlers are supplied by HLLs, some by the user. The condition handlers in turn can determine the correct action to take in responding to the condition. Throughout this document, "condition manager" refers to the LE/370 condition manager whereas "condition handler" refers to the user and language-specific condition handlers described below.

When a condition is detected, it is first communicated to the condition manager. The condition manager can then pass control to condition handling routines which examine the cause of the condition and contribute to the process of correcting it. These routines include:

- **Debug tool**

  If you have invoked the debug tool using either the TEST run-time option (see "TEST|NOTEST" on page 250) or the CEETEST callable service (see "CEETEST — Invoke Debug Tool" on page 419) the debug tool will gain control when the condition occurs.

- **User condition handler**

  A queue of user condition handlers established by CEEHDLR can be associated with each stack frame in which they are established. For example, you could call CEEHDLR to establish for the same stack frame two user condition handlers, one that handles program checks and another that handles ABENDs. The CEEHDLR service dynamically requests the LE/370 condition manager to pass control to a user-supplied condition handling routine when a condition occurs.

  The user condition handlers can respond to a condition in any of the ways described in "Responses to Conditions."

  See "User-Written Condition Handler" on page 98 for a description of the syntax for user-written condition handlers.

- **Language-specific condition handler**

  Language-specific condition handlers are provided by the HLL libraries and are automatically associated with a stack frame when a routine or procedure is entered. They enforce the language condition handling semantics for that stack frame. The language semantics depend on the language of the routine that obtained the stack frame.

  A language-specific condition handler can respond to a condition in any of the ways described in "Responses to Conditions."

  Language-specific condition handlers must "percolate" (described in the following section) all unknown conditions. An unknown condition is one for which the HLL has no defined action.

  C/370 Considerations: The C/370 language-specific condition handler may invoke for some events other user-written routines to handle conditions such as C/370 user `signal()` handlers. User `signal()` handlers are C/370-specific and are distinct from user-written condition handlers established using the CEEHDLR callable service. They are described in detail in "Using Signals" on page 91.

Responses to Conditions

A language-specific condition handler or a queue of user condition handlers is invoked by the condition manager at each stack frame to handle a condition. These language and user condition handlers respond to a condition in one of the following ways:

Resume A resume occurs when a condition handler determines that the condition was handled and normal application execution should resume. Program execution resumes at the instruction pointed to by the "resume cursor," usually the instruction immediately following the point at which the error occurred.

Note that language-specific and user-written condition handlers can modify the position of the resume cursor using the CEEMRCR callable service (see "CEEMRCR — Move Resume Cursor Relative to Handle Cursor" on page 300).

Percolate Percolation of a condition occurs when a condition handler declines to handle the condition. The LE/370 condition manager can continue condition handling in one of two places:

- With the next condition handler in a queue of user-established condition handlers, or the language-specific condition handler associated with the current stack frame
- With the first condition handler in a queue of user-established condition handlers for the calling stack frame (that is, the stack frame associated with the routine that called the current routine in the application).

Note: HLL-specific condition handlers must percolate all unknown conditions.

Promote

A condition is promoted when a condition handler converts the condition into a new condition with a different meaning. A condition handler can promote a condition for a variety of reasons, including the condition handler's knowledge or lack of knowledge about the cause of the original condition. A condition can also be promoted to simulate conditions that would normally come from a different source.

Important! Promoting a condition changes the HLL condition handling semantics for your routine. Do *not* promote conditions unless you have a thorough understanding of the condition handling semantics of the language in which your routine is written.

After a condition is promoted, the LE/370 condition manager continues processing by invoking language-specific and user condition handlers to handle the promoted condition in one of the following locations:

- With the next (language-specific or user) condition handler associated with the current stack frame
- With the first user condition handler associated with the stack frame that called the routine that incurred the condition.

Condition Management Model

Under LE/370, a new stack frame is obtained every time a routine is entered. Associated with each stack frame are condition handlers with the following characteristics:

- There is a single language-specific condition handler logically associated with each stack frame.
- A LIFO queue containing zero or more user condition handlers is associated with each stack frame. Calls to CEEHDLR for a given routine insert user condition handlers into the queue for the stack frame associated with that routine. If routine A calls routine B, there is a new queue associated with the stackframe for routine B. You can register user condition handlers using CEEHDLR on a stackframe by stackframe basis.
- The LE/370 condition manager invokes the user condition handlers at each stack frame to handle the condition. The most recent user condition handler registered using CEEHDLR is the first to be invoked.
- User condition handlers can be unregistered with a call to the CEEHDLU service (see "CEEHDLU — Unregister User Condition Handler" on page 296).

- Condition handlers are automatically unregistered when the routine that established the user condition handler returns to its caller. When a stack frame is exited, all condition handlers associated with the stack frame—user as well as language-specific—are automatically unregistered. The user condition handler queue may be empty for a given stack frame.
- User condition handlers are invoked before the language-specific condition handler associated with each stack frame.

When a condition occurs, the LE/370 condition manager starts at the most recently activated stack frame and proceeds stack frame by stack frame towards older stack frames. The condition handlers at each stack frame may respond by percolating, promoting, or handling the condition.

Condition handling is complete if one of the condition handlers handles the condition. If no condition handler handles the condition, the LE/370 condition manager eventually reaches stack frame 0, the stack frame that precedes the stack frame for the first routine in the thread. A language-specific condition handler is then invoked to enforce default condition handling semantics. The language of this handler depends upon the language of the routine in which the condition occurred.

If the language-specific handler percolates the condition, default condition handling actions defined by LE/370 are taken, based on the severity of the condition.

Handle and Resume Cursors

The condition manager uses two cursors to keep track of locations as it traverses the stack, stack frame by stack frame:

- The **handle cursor** points at the current stack frame and the condition handler that is currently being invoked. It is moved by the condition manager when processing is complete for all condition handlers associated with the stack frame. Note that processing can be restarted with the first condition handler, or terminated by any condition handler associated with the stack frame. The location where it is moved depends on the actions that were taken by the condition handler, as described in "Responses to Conditions" on page 85.
- The **resume cursor** designates the point in the application where execution can be restarted if the condition is handled. Usually, this is the next sequential instruction after the point in the application where the condition occurred. You can move the resume cursor using the CEEMRCR callable service (see "CEEMRCR — Move Resume Cursor Relative to Handle Cursor" on page 300 for more information).

TRAP(ON) effects

The TRAP run-time option affects how the condition manager acts upon error conditions and program interrupts. When TRAP(ON) is specified or is a run-time option default, conditions or program interrupts are signaled to the condition manager. The user condition handlers and language-specific condition handlers can then be invoked to handle them. If the signaled condition is not handled, a non-zero return and feedback code is returned to the invoker of the enclave.

If TRAP(OFF) is in effect, the condition manager never sees program interrupts and ABENDs. Therefore, program interrupts and ABENDs do not undergo LE/370 condition management.

User and language-specific handlers are *always* invoked for conditions signaled by the software using the CEESGL callable service, regardless of the setting of TRAP.

The ABPERC run-time option allows you to specify which (if any) ABEND code should be percolated by the LE/370 condition manager. ABPERC is intended for use as a debugging tool that allows the application to execute with TRAP(ON).

See "TRAP" on page 252 and "ABPERC" on page 216 for more information about the TRAP and ABPERC run-time options.

Condition Handling Example

Figure 35 contains a diagram indicating how the LE/370 condition management model handles a condition in an application comprised of multiple routines that call one another.

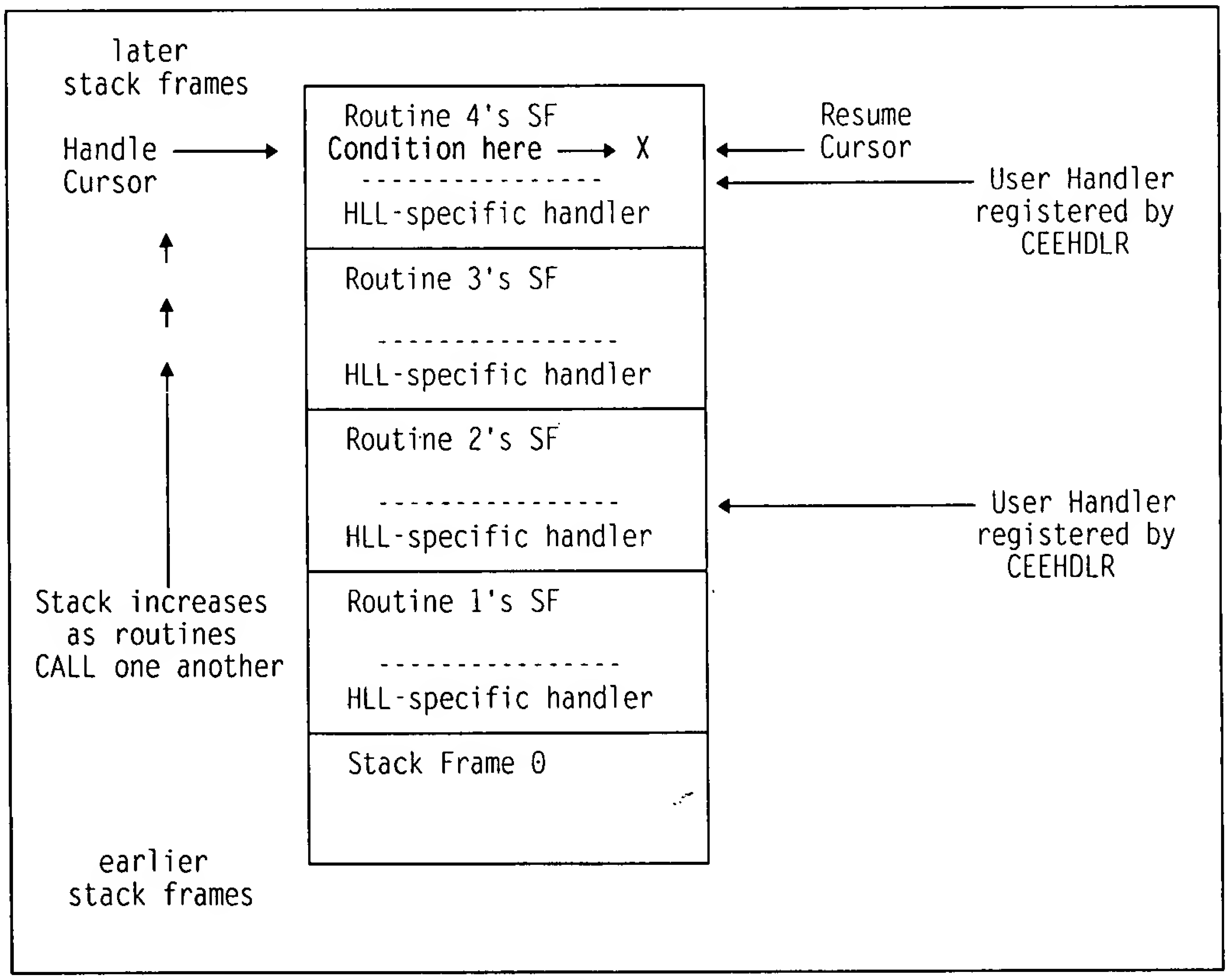


Figure 35. LE/370 Condition Handling Model

In the figure, the stack frames for the invoked routines in an application reside conceptually on a LIFO stack. The first routine is the main routine. It calls routine 2, which in turn calls routine 3, which calls routine 4. The condition occurred in routine 4 or was signaled in routine 4 by a call to CEESGL.

The **resume cursor** in Figure 35 points to the point just after where the error occurred. The **handle cursor** points at the condition handler that is invoked for the error in the stack frame associated with routine 4.

At the point when the condition occurs, the LE/370 condition manager takes the following steps:

1. The user condition handler registered by CEEHDLR for routine 4 is called. It accesses information about the error gathered by the condition manager and analyzes the condition. It can respond by taking one of the following actions: resume, promote, or percolate. Assume that it has no defined action for the condition and percolates it.
2. The language-specific condition handler for the stack frame is then called. It analyzes the condition to determine if it can handle the condition. Assume it also percolates the condition.
3. Because there are no more condition handlers for this stack frame, the LE/370 condition manager moves to the next stack frame, that for routine 3.

   The handle cursor now points at the user condition handler for the stackframe associated with routine 3. The resume cursor still points at the instruction just after where the condition occurred.
4. Because no user condition handler is registered for the stack frame for routine 3, the language-specific condition handler is given a chance to handle the condition. If it percolates the condition, the user condition handler at the next stack frame, that for routine 2, is invoked to handle it.
5. Assume that a user condition handler for routine 2 recognizes the condition and handles it. Control passes to the user condition handler routine, which handles the condition, and requests a resume.
6. After the routine completes, execution of the application resumes at the instruction where the resume cursor points.

If all of the condition handlers for the stack frames in Figure 35 percolated the condition, the LE/370 condition manager would eventually reach stack frame 0. The HLL-specific condition handler (based on the language of the routine that incurred the condition) is invoked to provide default condition handling. If the HLL-specific condition handler does not handle the condition, LE/370-defined default actions, as indicated in Table 19, are applied. The table indicates LE/370 default responses that occur when a condition remains unhandled after all condition handlers choose to percolate it.

Table 19. Default Responses to Unhandled Conditions

Severity of condition	Condition originated using CEESGL callable service	Condition originated using all other sources
0 (Informative message)	Return the unhandled condition.	Resume without issuing message.
1 (Warning Message)	Return the unhandled condition.	For COBOL routines, resume and issue the message. For non-COBOL routines, resume without issuing message.
2 (Program terminated in error)	Return the unhandled condition.	Terminate the thread (that is, your application terminates). Message issued if TERMTHDACT(MSG) is specified.
3 (Program terminated in severe error)	Return the unhandled condition.	Terminate the thread. Message issued if TERMTHDACT(MSG) is specified.
4 (Program terminated in critical LE/370 error)	Terminate the thread. Message issued if TERMTHDACT(MSG) is specified.	Terminate the thread. Message issued if TERMTHDACT(MSG) is specified.

In the current release of LE/370, only enclaves with single threads are supported. Thus, when the thread terminates, the enclave terminates.

TERMTHDACT run-time option: You can use the TERMTHDACT run-time option to set the kind of information you receive after your application terminates in response to a severity 2, 3, or 4 condition. For example, you can specify that a message or dump should be generated if the application terminates. For more information see the LE/370 *Language Environment/370 Debugging and Run-time Messages Guide* and "TERMTHDACT" on page 249.

ERRCOUNT Run-time Option

The ERRCOUNT option allows you to specify the number of errors that are tolerated during the execution of an enclave. Each condition of severity 2 or above, regardless of its origin, increments the error count by one. If the error count exceeds the limit, the condition manager terminates the enclave with ABEND code 4091 and reason code 11.

See "ERRCOUNT" on page 229 for more information.

Language-specific Condition Handling Semantics

The models for language-specific condition handling fall into two categories — the stackframe-based model and the Global Error Table (G.E.T.) model.

Current C Actions: C/370 employs a global error table to handle conditions. Upon initialization, C/370 defines certain actions that are to be taken when a condition is raised. C/370 users, however, can alter the action that the C/370 condition handler takes when a condition is raised by coding `signal()` function calls in their applications.

A G.E.T. determines the actions for handling conditions for the entire duration of application execution. You explicitly alter these actions by using signals. The actions defined by C/370 will apply to an entire application, not just a routine or block within an application. In contrast, condition handling can differ from stack frame to stack frame in the stack frame-based model previously described. Condition handling for one stack frame is not necessarily the same in the next stack frame on the stack. This distinction can affect how a condition is handled in your application, particularly if it is a mixed-language application comprised of a combination of COBOL and C/370 routines.

Current COBOL actions: COBOL provides some condition handling on a statement-by statement basis; for example the ON EXCEPTION clause of the CALL statement and the ON SIZE ERROR clause of the COMPUTE statement. For other conditions, COBOL generally reports the error, then ABENDs.

The following sections describe language-specific condition handling semantics for both C/370 and COBOL. A discussion of C-COBOL ILC condition handling semantics is also included.

C/370 Condition Handling Semantics

C/370 recognizes a number of errors that correspond directly to the errors that are detected by the hardware or the operating system, and some that are unique to C. All actions for condition handling are controlled by the contents of the C error option table. Table 20 lists the condition handling actions that are initially defined for C/370 applications.

Table 20. C/370 conditions and default system actions

C/370 condition	Hardware Exception / Origin	Default Action
SIGILL	operation exception privileged operation execute exception raise(SIGILL)	abnormal termination (return code=3000)
SIGSEGV	protection exception addressing exception specification exception raise(SIGSEGV)	abnormal termination (return code=3000)
SIGFPE	data exception fixed point divide decimal divide exponent overflow floating point divide raise(SIGFPE)	abnormal termination (return code=3000)
SIGABRT	abort() function raise(SIGABRT)	abnormal termination (return code=2000)
SIGTERM	termination request raise(SIGTERM)	normal termination (return code = 0)
SIGINT	attention condition	ignore the condition
SIGUSR1	user-defined condition	ignore the condition
SIGUSR2	user-defined condition	ignore the condition

When the C/370 function `raise()` is issued for any of the conditions listed in Table 20, a corresponding LE/370 condition is automatically raised by a call to the CEESGL callable service. Any of these LE/370 conditions (EDC6000 through EDC6008) can be handled by a user-written condition handler registered using the CEEHDLR service. For detailed descriptions of conditions EDC6000 through EDC6008, see the *IBM SAA AD/Cycle C/370 Programming Guide* and the *Language Environment/370 Debugging and Run-time Messages Guide.*

Using Signals

C/370 allows you to alter the actions that will be taken for any given signal using the C/370 `signal()` function call to handle error conditions. Signals are conditions that are not necessarily reported during application execution. For example, `SIGFPE` is the signal used to represent erroneous arithmetic operations such as a division by zero. Signals can be reported as a result of a machine interrupt or if you explicitly request to report a signal using the *raise* function.

Terminology: Signal is defined differently under C/370 and LE/370. You need to understand this distinction to avoid confusing the two meanings of the term.

- Using LE/370 services, you *register* a condition handler by CEEHDLR and *raise* a condition using CEESGL.
- Using C/370 functions, you *register* a condition handler using the `signal()` function and *raise* a condition using the `raise()` function.

You can think of signal as the C/370 term for an LE/370 condition. To simplify the following discussion, the term "condition" will be used in place of "signal".

You can use the `signal()` function to do the following:

- Ignore the condition completely. You can do this by specifying `SIG_IGN` in the `signal()` function. When the action for the condition is "ignore", the condition is considered to be **disabled**.
- Reset condition handling to the system default as shown in Table 20 on page 91. Actions for handling the condition are implicitly reset to the system default when the condition is reported. However, you can also explicitly reset handling actions to the default by specifying `SIG_DFL` in the `signal()` function.
- Call a user-written C function to handle the condition. You can do this by specifying, in the `signal()` function, a pointer to the user-written function that is to be called when the condition occurs.

  The user-written function specified in `signal()` is invoked from the C/370-specific condition handler. It is therefore given a chance to handle a condition only after any user-written handler established using CEEHDLR is invoked.

For more information about using the `signal()` and `raise()` functions, see the *IBM SAA Common Programming Interface C Reference—Level 2.*

C/370 Condition Handling Actions

Given the calling sequence of Routine 1 calling Routine 2 that in turn calls Routine 3, Routine 3 is the *owning* stack frame if a condition occurs while Routine 3 executing.

If a condition occurs while a C/370 routine is executing (that is, C/370 is the owning stack frame), then the sequence of actions that the C/370 condition handler performs is as follows:

1. If the action defined for the condition is to ignore it (that is, if `SIG_IGN` is specified as a parameter in the `signal()` function for the condition), the condition is disabled. Execution continues at the next sequential instruction after the point where the condition occurred.
2. If `SIG_IGN` is not specified for the condition, the debug tool is given a chance to handle the condition.
3. If the debug tool does not handle the condition, then the user condition handlers registered by CEEHDLR for that stack frame are invoked in LIFO order.
4. If all of these handlers percolate the condition, and if you have specified by the `signal()` function that control is to pass to a function when the condition is reported, then that function is invoked.

5. If the function handles the condition, control returns to the routine where the condition occurred. If the function *cannot* handle the condition, it may force a termination by issuing `exit()`, `abort()`, or may issue a `longjmp()`. The condition is percolated to the next user condition handler registered using CEEHDLR or the language-specific condition handler associated with the next stack frame *only* if `SIG_DFL` has been specified.
6. If the condition is percolated back to stack frame 0, the C/370-specific condition handler gains control and performs default condition handling actions for the signal as defined in Table 20 on page 91.

If a C/370 stack frame is *not* the owning stack frame and a C/370 routine is encountered on the stack, then the set of possible condition-handling actions that the C/370 handler can perform are:

- A user-written C/370 function is invoked if this function has been registered for the reported condition using `signal()`.
- In all other cases the C/370 condition handler will percolate the condition to the next condition handler.

C/370 Condition Handling Example

Figure 36 contains a simple example of a C/370 application where `y = a/b` is a mathematical operation. `signal (SIGFPE, c_handler)` is a signal invocation that registers the routine `c_handler()` and gives it control if a floating point divide exception occurs.

```
#include <stdio.h>
#include <signal.h>


int main(void) {
      .
      .
      .
   signal (SIGFPE, c_handler);
      .
      .
      .
   y = a/b;
      .
      .
}
void c_handler();
      .
    printf("handled SIGFPE\n");
      .
      .
    return;
}
```

Figure 36. C/370 Condition Handling Example

If b = O, a floating point condition occurs. In this case the condition manager determines the following:

- Is the condition disabled (is SIG_IGN specified for the condition)?
- Can the debug tool handle the condition?
- Can any user condition handler registered by CEEHDLR for that stack frame handle the condition?

If the answer is no for all of the above, the condition manager gives the C language-specific handler control. This handler in turn invokes c_handler() as specified in the signal() function in Figure 36 on page 93. Control is then returned to the instruction following the one that caused the condition.

COBOL Condition Handling Semantics

If a condition originates in an owning stack frame associated with a COBOL routine or block, the COBOL condition handler does not handle the condition until every user condition handler at every stack frame has been interrogated. The COBOL condition handler handles the condition only when it has been percolated to stack frame 0. Note that the COBOL condition handler *only* handles conditions that have a facility id of "IGZ".

At stack frame 0, the COBOL language-specific handler will handle the condition based on its severity (see Table 19 on page 89 for an explanation of severity codes and their meaning under LE/370). If the condition severity is 1, a message describing the condition is issued to the destination specified in the MSGFILE run-time option, and the COBOL condition handler resumes. If the severity is 2 or above, the COBOL condition handler percolates the condition, causing the LE/370 default condition handling actions to occur.

Resuming Execution After an "IGZ" Condition Occurs

There are some circumstances when a user condition handler may not issue a "resume" after a COBOL-specific "IGZ" condition of severity 2 or higher occurs. Two of these scenarios are illustrated in Figure 37 and Figure 38 on page 95, respectively.

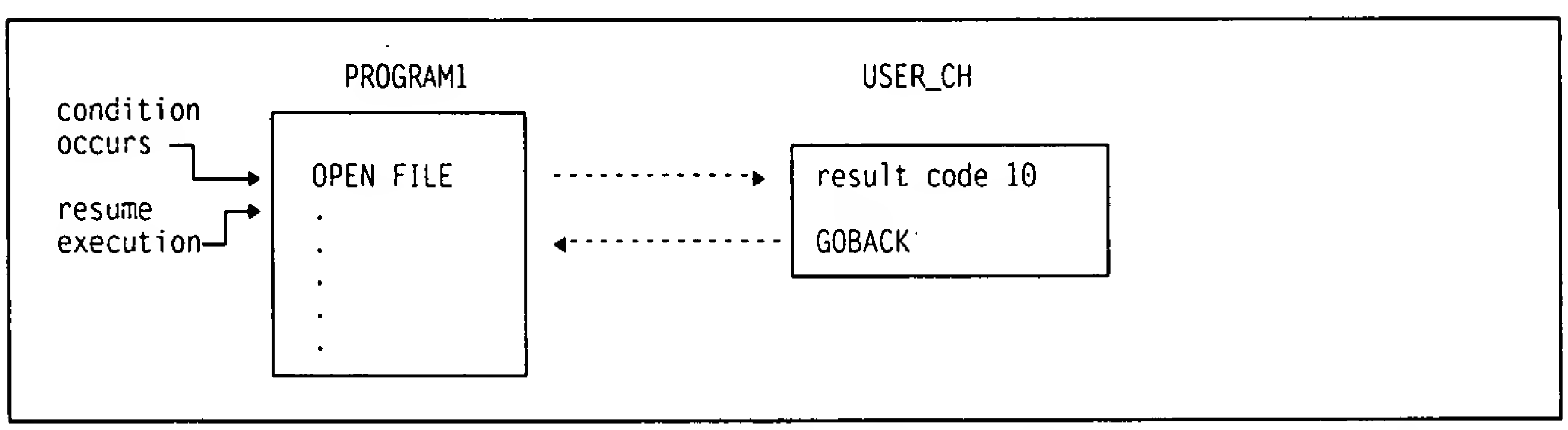


Figure 37. Resuming Execution After an "IGZ" Condition of Severity 2 or Above. Note that this is not permitted under COBOL.

In Figure 37, the COBOL routine PROGRAM1 encounters a severity 2 or higher condition as it attempts to open a file. Since the user-written condition handler USER_CH, was registered for PROGRAM1 using CEEHDLR, it is invoked to handle the condition. In this example, the USER_CH user handler that is provided cannot respond by resuming execution after the point in PROGRAM1 where the condition occurred (that is, you cannot code a *result_code* 10 in the USER_CH — see "User-Written Condition Handler" on page 98). Severity 2 or higher conditions in this case should be percolated.

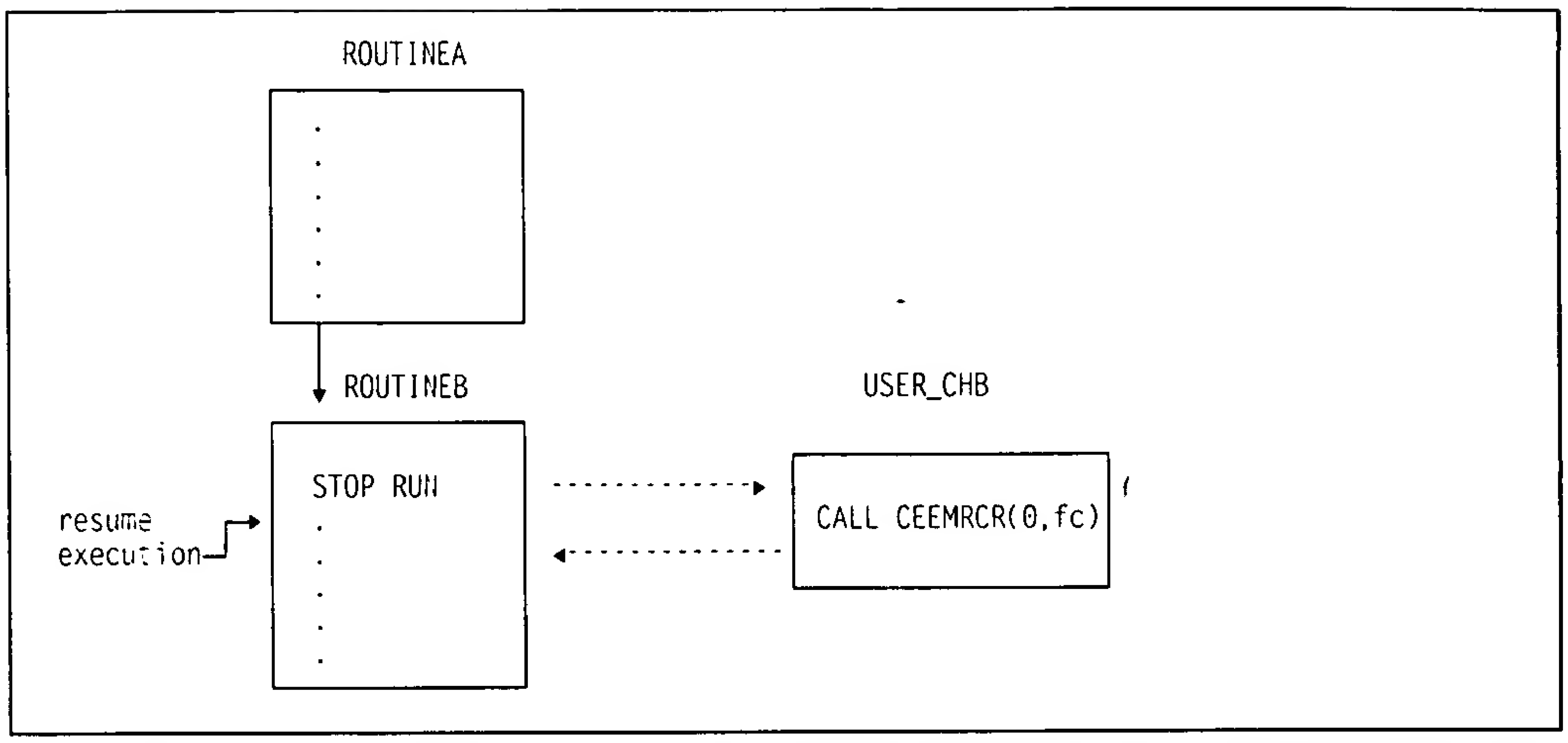


Figure 38. Moving the Resume Cursor after a STOP RUN Statement. Note that this is not permitted under COBOL.

You also cannot resume after a COBOL STOP RUN statement by moving the resume cursor. In Figure 38, a STOP RUN is issued in ROUTINEB which raises the Termination_Imminent condition. Control then passes to the user-written condition handler established for it, USER_CHB. USER_CHB then calls the CEEMRCR callable service ("CEEMRCR — Move Resume Cursor Relative to Handle Cursor" on page 300) with a 0 *type_of_move*, meaning move the resume cursor to the point in ROUTINEB just after the STOP RUN statement. ROUTINEB continues running at this point. This violates the standard definition of STOP RUN, normally the last statement to execute in the routine in which it is coded. Therefore, this type of resume is *not* permitted under LE/370.

Handling Fixed-Point and Decimal Overflow Conditions

The ON SIZE ERROR clause continues to be invoked by COBOL to handle fixed-point and decimal overflow conditions, regardless of whether these conditions are enabled by LE/370. However, ON SIZE ERROR is *not* invoked to handle these conditions if :

- The conditions are enabled by LE/370 using the CEE3SPM callable service (see "Using XUFLOW and CEE3SPM to Enable and Disable Hardware Conditions" on page 98), and
- The LE/370 condition manager is instructed to ignore program interrupts by the TRAP(OFF) run-time option setting.

The condition is instead percolated back to the operating system.

ILC Condition Handling Semantics

The following discussion of condition handling in an ILC application assumes that no user condition handlers are registered by CEEHDLR at any stack frame in the application.

C/370 routine calling COBOL routine: Given an ILC application where C_funct, a C/370 routine, calls COBOL_sub, a COBOL routine, and a condition occurs in COBOL_sub, the following sequence of actions can take place:

1. The COBOL handler associated with that stack frame percolates the condition to routine C_funct's stack frame, the next one on the stack.

2. At the C_funct stack frame, one of the following actions occurs:

   - If a C/370 signal handler for the condition raised in routine COBOL_sub has been registered by the `signal()` function, then that handler is invoked to handle the condition. Once control is returned from the user-written signal handler, the condition is considered **handled**. Execution resumes at the instruction following the one that caused the condition.
   - If no C/370 signal handler was registered by the `signal()` function, the condition is percolated.

3. If the condition is percolated to stack frame 0, LE/370 default condition handling actions occur, as illustrated in Table 19 on page 89.

COBOL routine calling C/370 routine: Given an ILC application where COBOL_main, a COBOL routine, calls C_funct, a C/370 routine, and a condition occurs in C_funct, the sequence of actions that can occur are:

1. The C/370 condition handler determines if it should handle the condition. If `SIG_IGN` is specified in the `signal()` function call, C/370 does not handle the condition. Execution then resumes at the next sequential instruction after the point where the condition occurred. If `SIG_IGN` is not specified, the condition is enabled and the C/370 condition handler continues processing the condition.
2. At the stack frame for the C/370 routine, one of the following occurs:

   - If a C/370 signal handler for the condition raised in routine C_funct has been registered by the `signal()` function, then that handler will be invoked to handle the condition. Once control is returned from the user-written signal handler, the condition is considered **handled**. Execution resumes at the instruction following the one that caused the condition.
   - If you have not registered a signal handler or specified `SIG_IGN` in the signal () function call, the condition is percolated to the next condition handler.

3. If the condition is percolated to stack frame 0, LE/370 default condition handling actions occur.

Additional ILC Condition Handling Considerations

- The `signal()` function can only register C/370 routines as user-written signal handlers. However, the C/370 routine may call other COBOL routines.
- C/370 user-written condition handlers established by the `signal()` function can be fetched (by the `fetch()` function).
- COBOL/370 user-written condition handlers established using CEEHDLR can be statically and dynamically loaded. They can be the target of a CALL or CANCEL statement. However, if the user-written condition handler is cancelled, it is no longer a valid address and any operation on it is illegal.
- CEEHDLR can register handlers written in a language different from the language of the routine that calls CEEHDLR.
- User-written condition handlers registered via CEEHDLR must be naturally reentrant, that is, contain no external data.

Nested Conditions

A *nested condition* is one that occurs within a HLL-specific or user-written condition handler invoked to handle a condition. When conditions occur during the condition handling process, the handling of the current condition is suspended and further action is taken based upon the state of the condition handling.

By default, nested conditions are not allowed under LE/370. However, the CEE3CNC callable service can be invoked to allow nested conditions to occur. In addition, nested conditions are allowed under C/370 in user-written handlers registered by the `signal()` function.

If a nested condition is allowed within a user-registered condition handler, the LE/370 condition manager begins handling the most recently raised condition. Once the most recently raised condition is properly handled, execution begins at the instruction pointed to by the resume cursor, the instruction following the point where the condition occurred. If a user condition handler is registered using CEEHDLR within another user condition handler, nested conditions are handled by the most recently registered condition handler.

For more information about CEE3CNC, see "CEE3CNC —Allow Nested Conditions" on page 315.

Nested Conditions in ILC applications.

You must take special care when dealing with nested conditions in ILC applications. For example, the following scenario can cause your application to ABEND:

1. A nested condition occurs within a COBOL/370 user-written condition handler (COBOL_uhdlr)
2. The COBOL/370 user condition handler calls another user-written condition handler established using CEEHDLR to handle the nested condition
3. The user-written condition handler percolates the condition.

In this scenario, the condition can be percolated back to the stack frame where the original condition occurred. Since condition handling actions for the routine where the condition originally occurred include calling COBOL_hdlr, COBOL_hdlr may be recursively entered. This is not permitted under COBOL/370, and your application will ABEND.

A rule of thumb is to ensure that COBOL/370 user-written condition handlers which call other user-written handlers do *not* regain control.

LE/370-issued ABENDs

LE/370 issues ABENDs for some errors. For these errors, normal condition handling does not occur. Instead, the condition manager terminates the process without language-specific or user condition handlers being invoked. All enclaves in the process are therefore terminated.

Note that you can also specify in the assembler user exit a list of ABEND codes that the LE/370 condition manager will percolate. Both system ABENDs and user ABENDs may be specified. See Chapter 27, "Advanced User Exit Topics" on page 173 for more information.

For a list of LE/370-issued ABENDs and information about using ABPERC to debug your application, see the LE/370 *Language Environment/370 Debugging and Run-time Messages Guide.*

Using XUFLOW and CEE3SPM to Enable and Disable Hardware Conditions

Bits 20 through 23 of the *Program Status Word* (PSW) can be set to enable or disable hardware interrupts such as:

- FIXED POINT OVERFLOW
- DECIMAL OVERFLOW
- EXPONENT UNDERFLOW
- SIGNIFICANCE.

LE/370 provides the CEE3SPM callable service to replace Assembler language routines that manipulate the PSW to enable or disable these conditions. The XUFLOW run-time option is also provided to enable or disable the EXPONENT UNDERFLOW condition.

Important! CEE3SPM and XUFLOW are provided by LE/370 to replace Assembler routines that manipulate the program mask of the PSW. Both may change the condition handling semantics of the HLL or HLLs of your application. Therefore, they are *not* intended for general usage.

C/370 Considerations: C/370 language semantics are undefined when XUFLOW(ON) is specified.

For more information, see "XUFLOW" on page 254 and "CEE3SPM — Query and Modify LE/370 Hardware Condition Enablement" on page 318.

User-Written Condition Handler

This section defines the interface that you use when you code a call to a user-written condition handler. Use the following syntax when making your call.

Syntax

```
▸▸——condition_handler——(—C_CTOK—,—token—,——result_code—,———▸

▸———new_condition—)——————————————————————————————————————▸◂
```

C_CTOK (input)
a 12-byte condition token that identifies the current condition being processed.

token (input)
a 4-byte integer.

Specifies the token that was passed into the LE/370 condition manager when this condition handler was registered by a call to the CEEHDLR callable service. See "CEEHDLR — Register User Condition Handler" on page 294 for more information.

result_code (output)
a 4-byte integer.

Result_code contains instructions about responses that the user-written condition handler wants the LE/370 condition manager to make when processing the condition.

Result_code is passed by reference. The following responses are valid:

Response	Value	Action
resume	10	Resume at the resume cursor (condition has been handled).
percolate	20	Percolate to the next condition handler.
	21	Percolate to the first user condition handler for the stack frame that is before the one to which the handle cursor points. (This can skip a language-specific condition handler for this stack frame as well as the remaining user written condition handlers in the queue for this stack frame.)
promote	30	Promote to the next condition handler.
	31	Promote to the stack frame before the one to which the handle cursor points. (This can skip a language-specific condition handler for this stack frame as well as any remaining user condition handler in the queue at this stack frame.)
	32	Promote and restart condition handling at the first condition handler of the stack frame of the handle cursor.

new_condition (output)
a 12-byte condition token

The new_condition token represents the promoted condition. This field is used only for *results* values of 30, 31, and 32, denoting **promote**.

Usage Notes

1. It is invalid to promote a condition without returning a new condition token.
2. The user condition handlers are registered by CEEHDLR. See "CEEHDLR — Register User Condition Handler" on page 294 for more information.
3. The user condition handlers are explicitly unregistered using CEEHDLU. See "CEEHDLU — Unregister User Condition Handler" on page 296 for more information.
4. The user condition handlers are automatically unregistered when the routines terminate due to a return, a GOTO out of block, or by moving the resume cursor.
5. Recursion is allowed if a handler is registered within a handler, and a call has been made to the CEE3CNC callable service allowing nested conditions (see "CEE3CNC —Allow Nested Conditions" on page 315).

Callable Services Available to Handle Conditions

In addition to the services already discussed in this chapter, LE/370 offers other callable services to assist you with other aspects of condition handling. These include:

- "CEEGQDT — Retrieve Q_Data_Token" on page 291
- "CEEITOK — Return Initial Condition Token" on page 298
- "CEE3GRN —Get Name of Routine that Incurred Condition" on page 316.

Examples

This section contains examples of how to use LE/370 services to handle conditions in the run-time environment.

Condition Handling Without a Registered User Condition Handler

COBOL/370 Example: The following three figures provide a simple illustration of how condition handling works under LE/370 when, for example, a zero divide condition occurs in a COBOL/370 application. The COBOLA routine in Figure 39 calls COBOLB in Figure 40 on page 101, which in turn calls the COBOLC routine in Figure 41 on page 101. A zero divide condition occurs in COBOLC.

Since there is no user-written condition handler registered for COBOLC or any of the other COBOL routines, the condition is percolated down the stack towards stack frame 0. Because the condition originated in an owning stack frame associated with a COBOL routine, the COBOL language-specific condition handlers percolate the condition to stack frame 0. The zero divide condition has a severity of 3, so the COBOL language-specific condition handler percolates the condition to LE/370. The LE/370 default response is to terminate the application and issue a message if TERMTHDACT(MSG) was specified.

```
CBL C,RENT,Q,NOOPTIMIZE,LIST,NODYNAM,TEST(SYM)
    *****************************************************************
    *                                                               *
    *  Demonstrate a failing COBOL program with multiple active *
    *  routines on the stack. The call sequence is as follows:  *
    *                                                               *
    *  COBOLA --> COBOLB --> COBOLC (which causes a zero divide)*
    *                                                               *
    *****************************************************************
     IDENTIFICATION DIVISION.
     PROGRAM-ID. COBOLA.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.
     01   Y  PIC 999 VALUE ZERO.
    *
     PROCEDURE DIVISION.
         DISPLAY "In COBOLA.".
         CALL "COBOLB" USING Y.
         GOBACK.
```

Figure 39. COBOLA routine

```
CBL C,RENT,Q,NOOPTIMIZE,LIST,NODYNAM,TEST(SYM)
    IDENTIFICATION DIVISION.
    PROGRAM-ID. COBOLB.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    1   X  PIC 999 VALUE ZERO.
    LINKAGE SECTION.
    1   Y  PIC 999.
   *
    PROCEDURE DIVISION USING Y.
        DISPLAY "In COBOLB.".
        MOVE Y TO X.
        CALL "COBOLC" USING X.
        GOBACK.
```

Figure 40. COBOLB routine

```
CBL C,RENT,Q,NOOPTIMIZE,LIST,NODYNAM,TEST(SYM)
    IDENTIFICATION DIVISION.
    PROGRAM-ID. COBOLC.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    LINKAGE SECTION.
    1   Y  PIC 999.
   *
    PROCEDURE DIVISION USING Y.
        DISPLAY "In COBOLC.".
        COMPUTE Y = 1 / Y.
        GOBACK.
```

Figure 41. COBOLC routine

C/370 Example: The following three examples provide a simple illustration of how condition handling works under LE/370 when, for example, a zero divide occurs in a C/370 application.

Since there is no user-written condition handler or signal registered for C370C or any of the other C/370 routines, the condition is percolated by the C/370 language-specific condition handler at each of the three stack frames associated with the routines until stack frame 0 is reached. At this point, the C/370 language-specific condition handler gains control and percolates the condition to LE/370. LE/370 will recognize the zero divide condition and terminate the application. A message, dump, or trace may be generated depending on the setting of TERMTHDACT ("TERMTHDACT" on page 249).

```
/**********************************************************************/
/* Demonstrate a failing C/370 program                                */
/* with multiple active routines                                      */
/* on the stack.  The call sequence is as follows:                    */
/*   C370A ---> C370B ---> C370C (which does a divide by zero)        */
/**********************************************************************/

#include <stdio.h>

int y = 0;
void C370B(void);

int main(void) {

  printf("In Program C370A\n");
  C370B();
}
```

Figure 42. C370A Routine

```
/**********************************************************************/
/* This routine is called to pass data forward to C370C.              */
/* C370C will then cause a zero divide.                               */
/**********************************************************************/

#include <stdio.h>

extern int y;
void C370C(int);

void C370B(void) {

  int x;
  printf("In Program C370B\n");
  x = y;
  C370C(x);
}
```

Figure 43. C370B Routine

```
/**********************************************************************/
/* This routine is called by C370B to generate a zero divide.         */
/**********************************************************************/

#include <stdio.h>


void C370C(int y) {

  printf("In Program C370C\n");
  y = 1/y;
}
```

Figure 44. C370C Routine

Condition Handling With User-written Condition Handler Registered

COBOL/370 Example: Figure 45 on page 103 and the following three examples provide an illustration of how user-written condition handlers can handle conditions such as a zero divide in a COBOL/370 application. The main routine EXCOND (Figure 46 on page 104) calls CEEHDLR to register the USRHDLR user-written condition handler (Figure 48 on page 106). It then calls the ZERODIV routine in which a zero divide condition occurs. The handle cursor, which first points at

ZERODIV's stack frame, moves down the stack to the USRHDLR condition handler, the first user handler established to handle conditions for the main routine's stack frame.

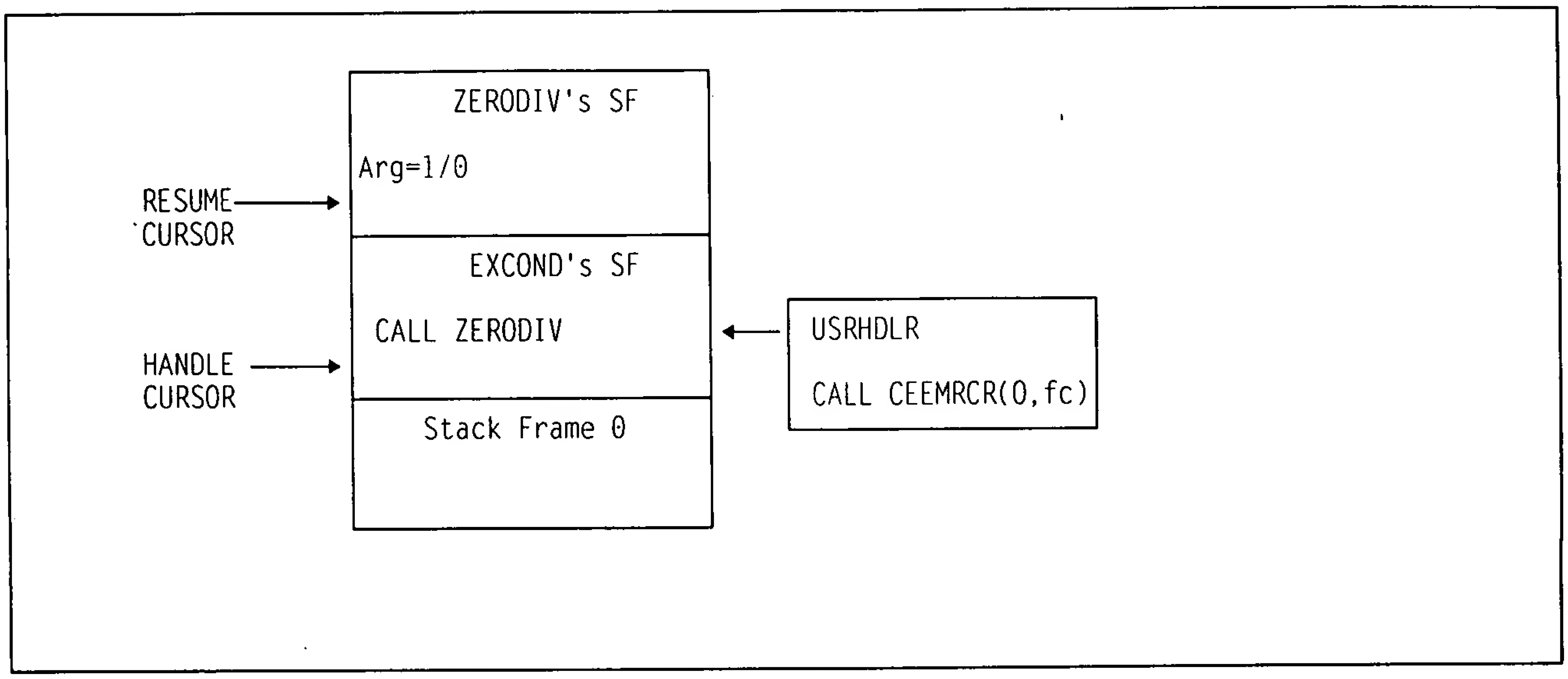


Figure 45. Handle and Resume Cursor Movement as a Condition is Handled

For zero divide conditions, USERHDLR issues a call to CEEMRCR ("Move Resume Cursor") with a 0 *type_of_move* meaning move the resume cursor to the call return point of the stack frame associated with the handle cursor. The call return point is the next instruction after the call to the ZERODIV routine. Execution resumes in EXCOND at this point. Note that a zero divide condition is the only type of program interrupt for which USRHDLR causes a resume; all other program interrupts are percolated to the next condition handler on the stack.

Note: For simplicity, the examples shown below do not include calls to some LE/370 services that might otherwise be useful for handling conditions in your application. For example, you might code in the USERHDLR routine a call to the CEE3GRN callable service in order to get the name of the routine that incurred the condition. In addition, after calls to any LE/370 service, it is good practice to check the condition token returned to the calling routine in order to determine if the LE/370 service completed successfully.

```
CBL C,RENT,Q,LIST,NODYNAM,TEST(SYM),NOOPT
*****************************************************************
*                                                               *
*  EXCOND                  |-> DIVZERO                          *
*  - register handler   |    - force a zero divide              *
*  - call DIVZERO      --|                                      *
*     ==> "resume point"                                        *
*  - unregister handler                                         *
*                              USRHDLR:                         *
*                              - if zero divide                 *
*                                - move resume cursor           *
*                                - resume at "resume point"     *
*                                                               *
*****************************************************************

*****************************************************************
**          I D         D I V I S I O N                       ***
*****************************************************************
 Identification Division.
 Program-id.    EXCOND.
 Author.        JONES.
 Installation.  IBM - Santa Teresa Laboratory.
*****************************************************************
**     E N V I R O N M E N T       D I V I S I O N            ***
*****************************************************************
 Environment Division.
 Configuration Section.
 Input-Output Section.
 File-control.
*****************************************************************
**          D A T A     D I V I S I O N                       ***
*****************************************************************
 Data Division.
 File Section.
 Working-storage Section.
 01   fbcode.
  02   fb-severity      pic 9(4) Binary.
  02   fb-detail        pic X(10).
 77   divisor    pic S9(9) Comp.
**
**  Declares for condition handling
**
 77   token      pic X(4).
 77   pgmptr usage is PROCEDURE-POINTER.
 01   CEE000 PIC X(12).
   88 fb-ok value x"000000000000000000000000".
 Linkage Section.
*****************************************************************
**          P R O C     D I V I S I O N                       ***
*****************************************************************
 Procedure Division.
 PARA-CND01A.
**
** Register a user condition handler.
**
     Set pgmptr to entry "USRHDLR".
     Move ZERO  to token.
     Call "CEEHDLR" using pgmptr token fbcode.
     Display "EXCOND: Registered USR-HDLR.".
```

Figure 46 (Part 1 of 2). EXCOND routine (COBOL/370)

```
**
** Call DIVZERO to force a zero divide and drive USRHDLR
**
    Move 00 to divisor.
    Call "DIVZERO" using divisor.
    Display "EXCOND: Resumption after DIVZERO.".
**
** Unregister the user condition handler.
**
    Call "CEEHDLU" using pgmptr fbcode.
    Display "EXCOND: Unregistered USRHDLR.".
     Goback.
 End program EXCOND.
```

Figure 46 (Part 2 of 2). EXCOND routine (COBOL/370)

```
CBL C,RENT,Q,LIST,NODYNAM,TEST(SYM),NOOPT
    *****************************************************************
    **          I D        D I V I S I O N                       ***
    *****************************************************************
     Identification division.
     Program-id. DIVZERO.
     Environment Division.
    *****************************************************************
    **          D A T A    D I V I S I O N                       ***
    *****************************************************************
     Data division.
     Working-storage section.
     Linkage section.
     1   Arg pic S9(9) Comp.
    *****************************************************************
    **          P R O C    D I V I S I O N                       ***
    *****************************************************************
     Procedure division using Arg.
         Display "  DIVZERO: Starting.".
         Compute Arg = 1 / Arg.
         Display "  DIVZERO: Returning to its caller.".
         Goback.
     End program DIVZERO.
```

Figure 47. DIVZERO routine (COBOL/370)

```
*****************************************************************
* USRHDLR                                                       *
*                                                               *
*                                                               *
*****************************************************************
****************************************************************
**          I D        D I V I S I O N                       ***
****************************************************************
Identification division.
Program-id. USRHDLR.
Environment division.
****************************************************************
**          D A T A    D I V I S I O N                       ***
****************************************************************
Data division.
Working-storage section.
1 Misc-Variables.
  02  move-type-0       pic s9(9) binary value zero.
  02  move-type-1       pic s9(9) binary value 1.
01   Feedback.
  02  Fb-severity       pic 9(4)  binary.
  02  Fb-detail         pic X(10).
*
Linkage section.
****************************************************************
*                                                              *
*  Declare the first 8 bytes of the condition tokens for S/370 *
*  program interrupt codes 0C1 thru 0CF.                       *
*                                                              *
****************************************************************
1 Current-condition.
  2 filler            pic x(8).
  88 CEE341 value x"00030C8159C3C5C5".
  88 CEE342 value x"00030C8259C3C5C5".
  88 CEE343 value x"00030C8359C3C5C5".
  88 CEE344 value x"00030C8459C3C5C5".
  88 CEE345 value x"00030C8559C3C5C5".
  88 CEE346 value x"00030C8659C3C5C5".
  88 CEE347 value x"00030C8759C3C5C5".
  88 CEE348 value x"00030C8859C3C5C5".
  88 CEE349 value x"00030C8959C3C5C5".
  88 CEE34A value x"00030C8A59C3C5C5".
  88 CEE34B value x"00030C8B59C3C5C5".
  88 CEE34C value x"00030C8C59C3C5C5".
  88 CEE34D value x"00030C8D59C3C5C5".
  88 CEE34E value x"00030C8E59C3C5C5".
  88 CEE34F value x"00030C8F59C3C5C5".
  2 filler            pic x(4).
**
1 Token pic x(4).
**
1 Result-code pic s9(9) binary.
  88 resume       value +10.
  88 percolate    value +20.
  88 perc-sf      value +21.
  88 promote      value +30.
  88 promote-sf   value +31.
1 New-condition  pic x(12).
****************************************************************
```

Figure 48 (Part 1 of 2). USRHDLR routine (COBOL/370)

```
      **          P R O C     D I V I S I O N                  ***
      ****************************************************************
       Procedure division
           using current-condition token result-code new-condition.
 *
           Display ">>> USRHDLR: Entered User Handler ".
              If CEE349 then
                 Call "CEEMRCR" using move-type-0 feedback
                 Set resume to true
                 Display ">>> USRHDLR: Resuming execution"
               Else
                 Set percolate to true
                 Display ">>> USRHDLR: Percolating it"
               End-if
              .
           Goback.
       End program USRHDLR.
```

Figure 48 (Part 2 of 2). USRHDLR routine (COBOL/370)

C/370 Example: The discussion of the previous COBOL/370 example applies to the example shown in Figure 49.

```
/****************************************************************/
/*                                                              */
/*  MAIN                    |-> DIVZERO                         */
/*  - register handler   |    - force a zero divide             */
/*  - call DIVZERO     --|                                      */
/*     ==> "resume point"                                       */
/*  - unregister handler                                        */
/*                            USRHDLR:                          */
/*                            - if zero divide                  */
/*                              - move resume cursor            */
/*                              - resume at "resume point"      */
/*                                                              */
/****************************************************************/
#include <leawi.h>
#include <stdio.h>
#include <string.h>

void usrhdlr(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
void divzero(int);
#define SUCCESS "\0\0\0\0"

int main(void) {

 _FEEDBACK fc;
 _INT4 divisor;
 _INT4 token;
 _ENTRY pgmptr;

 /* Register a user condition handler.                         */
 pgmptr.address = (_POINTER)&usrhdlr;
 pgmptr.nesting = NULL;
 token = 97;
 CEEHDLR (&pgmptr, &token, &fc);
 if (memcmp(&fc,SUCCESS,4) != 0) {
    printf("CEEHDLR failed with message number %d\n",fc.tok_msgno);
    exit(99);
 }
 printf("MAIN: Registered USRHDLR.\n");

 /* Call DIVZERO to force a zero divide and drive USRHDLR      */
 divisor = 0;
 divzero(divisor);
 printf("MAIN: Resumption after DIVZERO.\n");

 /* Unregister the user condition handler.                     */
 CEEHDLU (&pgmptr, &fc);
 if (memcmp(&fc,SUCCESS,4) != 0) {
    printf("CEEHDLU failed with message number %d\n",fc.tok_msgno);
    exit(99);
 }

 printf("MAIN: Unregistered USRHDLR.\n");
}

void divzero(int arg) {
 printf("  DIVZERO: Starting.\n");
 arg = 1 / arg;
 printf("  DIVZERO: Returning to its caller.\n");
}

/***********************************************************************/
/* usrhdlr will handle all DIVIDE BY ZERO conditions...all others will*/
/* get percolated.                                                     */
/***********************************************************************/

void usrhdlr(_FEEDBACK *cond,_INT4 *input_token, _INT4 *result,
             _FEEDBACK *new_cond) {
```

Figure 49 (Part 1 of 2). EXCOND routine (C/370)

```
  _INT4 move_type_0 = 0;
  _INT4 move_type_1 = 1;
  _FEEDBACK feedback;

  /* values for handling the conditions */
  #define resume      10
  #define percolate   20
  #define promote     30
  #define promote_sf  31

  /*****************************************************************/
  /*                                                               */
  /*  Declare the 2 byte values of the message numbers  for S/370 */
  /*  program interrupt codes 0C1 thru 0CF.                        */
  /*                                                               */
  /*****************************************************************/
  #define CEE3201     0x0C81
  #define CEE3202     0x0C82
  #define CEE3203     0x0C83
  #define CEE3204     0x0C84
  #define CEE3205     0x0C85
  #define CEE3206     0x0C86
  #define CEE3207     0x0C87
  #define CEE3208     0x0C88
  #define CEE3209     0x0C89  /* DIVIDE BY ZERO */
  #define CEE3210     0x0C8A
  #define CEE3211     0x0C8B
  #define CEE3212     0x0C8C
  #define CEE3213     0x0C8D
  #define CEE3214     0x0C8E
  #define CEE3215     0x0C8F

  printf(">>> USRHDLR: Entered User Handler \n");
  printf(">>> passed token value is %d\n",*input_token);

  /* check if the DIVIDE BY ZERO message (0C9) */
  if (cond->tok_msgno == CEE3209) {
      CEEMRCR (&move_type_0, &feedback);
      if (memcmp(&feedback,SUCCESS,4) != 0) {
         printf("CEEMRCR failed with message number %d\n",feedback.tok_msgno);
         exit(99);
      }
      *result = resume;
      printf(">>> USRHDLR: Resuming execution\n");
  }
  else {                        /* not DIVIDE BY ZERO */
        *result = percolate;
        printf(">>> USRHDLR: Percolating it\n");
  }
}
```

Figure 49 (Part 2 of 2). EXCOND routine (C/370)

Chapter 20. Message Handling

LE/370 permits communication between various routines within an enclave through the use of messages. Under LE/370, messages are issued in a common and consistent format across all LE/370-conforming HLLs.

Messages provide informational and diagnostic run-time information about your application that is either displayed on-line or collected in a file or data set whose name you specify. For example, if a condition is raised in your application, a condition token is generated. The condition token contains a unique id that LE/370 uses to create a detailed message associated with that condition.

A more detailed explanation of the relationship between messages and condition handling is contained in Chapter 18, "Communicating Conditions" on page 79. In addition, a complete listing of the run-time messages that can be generated while your application executes is contained in the *Language Environment/370 Debugging and Run-time Messages Guide.*

The following sections in this chapter describe:

- Delivering messages in LE/370
- Using National Language callable services to specify the format of messages that are delivered.

Delivering the Message

Run-time messages that are generated while your application is executing are directed to the ddname specified in the MSGFILE run-time option. You can associate this ddname with an I/O device using a FILEDEF statement under CMS, or in your JCL under MVS, as indicated in Table 21.

Table 21. Defining an I/O Device for a ddname

Operating System	I/O Device Defined Using	Refer to
CMS	The ddname of a file that you define using a FILEDEF statement	"Using FILEDEF to define input and output files" on page 54
MVS	The ddname of a data set that you specify in the JCL	"DD Statements for the Standard Data Sets" on page 39
TSO	The ddname of the data set that you specify using the ALLOCATE command	Chapter 12, " Creating a Load Module under TSO" on page 44

Dynamic Allocation of the MSGFILE ddname

If you do not explicitly associate a ddname with an I/O device, LE/370 dynamically allocates the MSGFILE ddname with IBM-supplied default attributes upon the first reference to MSGFILE in your application. The IBM-supplied default attributes are listed in Table 22 on page 111.

© Copyright IBM Corp. 1991

Table 22. IBM-supplied Default MSGFILE Destinations		
Operating System	**If no I/O device defined, LE/370 dynamically allocates to**	**IBM-supplied Default Attributes**
CMS	FILEDEF SYSOUT TERMINAL	LRECL 121 BLKSIZE 121 RECFM F NOCHANGE
MVS	SYSOUT=* [4]	LRECL 121 RECFM FBA BLKSIZE 121*100, if a non-terminal BLKSIZE 121, if a terminal
TSO	ALLOC DD(SYSOUT) DA(*)	LRECL 121 RECFM FBA BLKSIZE 121

You need to modify existing JCL and EXECs of pre-LE/370-conforming applications in order to define new ddnames for MSGFILE.

CICS Considerations: Message handling services provided by LE/370 under CICS are the same as those under batch, with the following exceptions:

- The MSGFILE run-time option is ignored under CICS. Messages for a CICS run-unit are directed to a CICS transient data queue named CESE. CESE is the only file to which LE/370 writes under CICS.
- Messages are prefixed by a Terminal_ID, a Transaction_ID, a date, and a time.
- The entire message record is preceded by an American National Standard control character to determine the format of the printing.
- Message records will be V-format.
- **COBOL considerations** — The COBOL/370 and VS COBOL II DISPLAY statement is not supported under CICS. The OS/VS DISPLAY, READY TRACE, and EXHIBIT statements are not supported under CICS.

See "Run-time Output" on page 129 for more information.

Using MSGFILE across Multiple Enclaves

You can specify the same MSGFILE across multiple (nested) enclaves. LE/370 coordinates the use of the same ddname across nested enclaves. If you specify different MSGFILE ddnames in each enclave, LE/370 honors each ddname.

Inserting Messages in Your Application

You can use messages in your application to show the location in an application and to display variable values. Each LE/370-conforming language provides a means to display messages. For example, COBOL offers the DISPLAY statement and C/370 provides the `printf` function.

[4] SYSOUT=* indicates the output is routed to the destination specified in the MSGCLASS option of the JOB card.

The LE/370 callable service CEEMOUT (see "CEEMOUT — Dispatch a Message" on page 330) enables you to dispatch a specified message to the LE/370 message file. You can use the MSGFILE run-time option to specify a ddname for the message file.

The CEEMOUT callable service can be used to return a message that indicates the status and progress of the program at particular points of execution.

COBOL Considerations: The COBOL output to the system logical output device is managed by LE/370. This output includes:

- DISPLAY SYSOUT (see "Using COBOL/370 DISPLAY and ACCEPT statements" on page 116 for syntax)
- READY TRACE (OS/VS COBOL only)
- EXHIBIT (OS/VS COBOL only).

By default, the output from the DISPLAY statement will be directed to SYSOUT, which is also the default for LE/370 run-time messages and reports. The destination of the DISPLAY messages can also be defined using the OUTDD compile-time option. The MVS dataset or CMS file where the messages are written depends on what combination of ddnames are specified for OUTDD and MSGFILE:

- If the ddname in OUTDD happens to match the ddname specified in the MSGFILE run-time option, the output is synchronized with the run-time messages and placed in the file (MVS dataset or CMS file) designated by the MSGFILE run-time option.
  - If the file designated by MSGFILE has not been defined (associated with an I/O device) when the output is delivered, LE/370 dynamically allocates the file with ddname and attributes as shown in Table 22 on page 111.
- If the ddname in OUTDD does *not* match the ddname specified in the MSGFILE run-time option, the DISPLAY/TRACE/EXHIBIT output is directed to the OUTDD ddname.
  - If the file designated by OUTDD has not been defined when the output is delivered, LE/370 dynamically allocates the file with ddname and attributes as shown in Table 22 on page 111.

The possible ddname specification combinations for OUTDD and MSGFILE and the locations where display output and run-time messages are routed are summarized in Table 23 on page 113.

Table 23. Run-time Message and DISPLAY Destinations for OUTDD and MSGFILE ddname Specifications under CMS		
ddname Specification	**FILEDEFs issued?**	**Destination**
MSGFILE(SYSOUT) OUTDD(SYSOUT)	Yes, for SYSOUT	Messages and DISPLAY data are routed to the destination defined for SYSOUT.
	No	LE/370 dynamically allocates FILEDEF SYSOUT TERM for messages and DISPLAY data.
MSGFILE(SYSOUT) OUTDD(ddname)	Yes, for SYSOUT	Messages are routed to the destination defined for SYSOUT.
	Yes, for ddname	DISPLAY data is routed to the destination defined for ddname.
	No	LE/370 dynamically allocates SYSOUT to FILEDEF SYSOUT TERM, the message destination. LE/370 dynamically allocates ddname to FILEDEF ddname DISK FILE ddname A1, the DISPLAY data destination.
MSGFILE(ddname) OUTDD(SYSOUT)	Yes, for ddname	Messages are routed to the destination defined for ddname.
	Yes, to SYSOUT	Display data is routed to the destination defined for SYSOUT.
	No	LE/370 dynamically allocates ddname to FILEDEF ddname TERM, the message destination. LE/370 dynamically allocates SYSOUT to FILEDEF SYSOUT DISK FILE SYSOUT A1, the DISPLAY data destination.

C/370 Considerations: C/370 makes a distinction between types of error output according to whether the output is directed to the MSGFILE, to the C/370 language "stderr", or to the C/370 language "stout" file:

Table 24. Output Destinations under C/370			
Type of Output	**Type of Message**	**Produced by**	**Default Destination**
MSGFILE output	LE/370 messages (CEExxxx)	LE/370 unhandled conditions	MSGFILE ddname
	C/370 language messages (EDCxxxx)	C/370 unhandled conditions	MSGFILE ddname
Stderr messages	Perror messages (EDCxxx)	Issued by a call to perror	MSGFILE ddname
	User output sent explicitly to stderr	Issued by a call to fprintf	MSGFILE ddname
Stdout messages	User output sent explicitly to stdout	Issued by a call to printf	Stdout

As indicated in Table 24, all stderr output is by default sent to the MGFILE destination, while stdout output is sent to its own destination. Stdout and stderr output can be redirected to a different ddname, file name, or to each other on the command line or by using freopen. For example, when stderr is redirected to stdout, both share the stdout destination. When stdout is redirected to stderr, both share the stderr destination.

Table 25 on page 114 illustrates the destination of MSGFILE, stderr, or stdout output when these have been redirected to a common destination (that is, when these outputs are *interleaved*).

Table 25. C/370 Interleaved Output

	Stderr not redirected	**Stderr redirected to destination other than stdout**	**Stderr redirected to stdout**
Stdout not redirected	Stdout to itself Stderr to MSGFILE	Stdout to itsef Stderr to its other destination	Both to stdout
Stdout redirected to destination other than stderr	Stdout to its other destination Stderr to MSGFILE	Stdout to its other destination Stderr to its other destination	Both to the other stdout destination
Stdout redirected to stderr	Both to MSGFILE	Both to the other stderr destination	When stderr and stdout are redirected to each other (this is not recommended), output from both is directed to whichever was specified first.

For more information about redirecting standard streams, see the *IBM SAA AD/Cycle C/370 Programming Guide.*

ILC Considerations: If C/370 is the main program in an ILC application, output from `printf` can be redirected to the LE/370 message file where it will be interspersed with output from the COBOL DISPLAY statement and output from LE/370. This is done by passing `1>&2` as a program argument to the main routine.

If COBOL is the main routine, there is no way to direct output from COBOL, C/370, and LE/370 to the same output file without using the CEEMOUT callable service.

Other Message Handling Run-time Options and Callable Services

- The MSGQ run-time option,"MSGQ" on page 235, specifies the number of ISIs (*Instance Specific Information*) that LE/370 allocates per thread. The ISI contains information about a particular occurrence or instance of a condition that is used to write a message about the condition to the MSGFILE or terminal.
- The CEEMGET callable service, "CEEMGET — Get a Message" on page 327, gets, formats, and stores a message in a buffer. The message can be sent to the MSGFILE or terminal, or manipulated in some other way by the calling routine.
- The CEEMSG callable service, "CEEMSG — Get, Format, and Dispatch a Message" on page 331, is used to write a message indicating the results of a call to an LE/370 service.
- The CEEMOUT callable service, "CEEMOUT — Dispatch a Message" on page 330, is used to write a message string that you supply to a specified destination.

Message Handling and National Language Support

LE/370 National Language Support (NLS) allows you to view run-time messages in your own national language. NLS allows you to select the most appropriate language variables such as language character set, left-to-right text, single-byte character set (SBCS), and double-byte character set (DBCS) for LE/370 messages.

LE/370 message handling provides services to support many NLS machine readable information requirements (MRI) such as:

- Message formatting
- Message delivery
- Normalization (removes adjacent shift-out, shift-in character in order to make DCBS strings as compatible as possible).

Specifying the National Language

The LE/370 message facility formats messages for any language known to LE/370. You can set the national language using the NATLANG run-time option (see "NATLANG" on page 235 for more information). The CEE3LNG callable service allows you to set or query the current national language setting, or add or remove a national language setting from the stack on a LIFO basis (see "CEE3LNG — Set National Language" on page 348 for more information). If the message text is not available for the current national language, the installation default is used instead.

LE/370 provides run-time messages for the following national languages:

- ENU — Mixed case English USA
- UEN — Uppercase American English
- JPN — Japanese.

Specifying the National Country Setting

The setting of the national "COUNTRY" controls the following default values for run-time messages as well as the date and time stamps in the reports generated by the RPTSTG option (storage report), RPTOPTS option (options report), and the CEE3DMP callable service (dump):

- Date Format
- Time Format
- Currency Symbol
- Decimal Separator Character
- Thousands Separator.

You can set the value of "COUNTRY" using the COUNTRY run-time option (see "COUNTRY" on page 224 for more information). You can also maintain more than one "COUNTRY" setting on a LIFO stack. The CEE3CTY callable service allows you to set or query the "COUNTRY" setting, or add or remove a "COUNTRY" setting from a stack on a LIFO basis (see "CEE3CTY — Set Default Country" on page 345).

Appendix G, "IBM-supplied Country Code Defaults" on page 469 contains a table of IBM-supplied default values for messages and reports based on the current COUNTRY setting.

Date and Time Services considerations: The current setting of COUNTRY may affect the format of the parameters input to and received from date and time services. Many date and time services allow you to specify a *picture_string* parameter that sets the format for inbound and outbound parameters. If *picture_string* is left null or blank, you receive the default format based on the current "COUNTRY" setting.

See Chapter 38, "Date and Time Services" on page 359 for more information.

Using COBOL/370 DISPLAY and ACCEPT statements

This section applies to COBOL applications only.

DISPLAY statement

The DISPLAY statement transfers run-time output to a logical output device. DISPLAY requires an associated DD statement in your MVS JCL or through a CMS FILEDEF command.

The DD statements are as follows:

Format 1

```
COBOL statement:

   DISPLAY  { identifier|literal } . . .

JCL statement:

   //SYSOUT DD applicable parameters
```

SYSOUT is an unblocked data set that has a default logical record length of 121 characters. This represents 1 byte for the carriage control character plus 120 bytes of user data from the COBOL application. For example:

```
//SYSOUT DD SYSOUT=A
```

You can specify an alternate logical record length, record format, or a blocked data set by using the DCB parameter.

To specify an alternate record length, use the subparameter of the DCB parameter as follows:

```
LRECL=record length+1
```

The extra character in LRECL allows for the carriage control character.

For example:

```
//SYSOUT     DD      SYSOUT=A,DCB=(LRECL=133,BLKSIZE=133)
```

To specify a blocked data set, for example:

```
//SYSOUT     DD      DSNAME=PRINTOUT,
//                   UNIT=SYSDA,
//                   DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605),
//                   VOLUME=SER=111111
```

If your application uses the SORT/MERGE feature, SYSOUT is the default error message data set for the DFSORT product, and a conflict can arise.

Format 2

```
COBOL statement:

   DISPLAY  { identifier|literal } . . . UPON SYSPUNCH

JCL statement:

   / SYSPUNCH  DD     applicable parameters
```

SYSPUNCH is an unblocked data set that has a logical record of 80 characters. For example:

```
//SYSPUNCH      DD      SYSOUT=B
```

However, you can specify a blocked data set by using the subparameters of the DCB parameter as follows:

```
RECFM=FB,BLKSIZE=n*80
```

where *n* is the blocking factor (and SYSPUNCH must be on a device where blocking is permitted).

Format 3

```
DISPLAY  { identifier|literal } . . . UPON mnemonic-name
```

A *mnemonic-name* can be used in a DISPLAY statement instead of a system-name. The *mnemonic-name* is defined in the Special-Names paragraph of the Environment Division.

When the UPON phrase is omitted, SYSOUT is the default device.

ACCEPT Statement

The ACCEPT statement transfers data, such as system information, to a specified data set. ACCEPT requires an associated DD statement unless:

The data is accepted from the console.
You use ACCEPT FROM with DAY, DATE, DAY-OF-WEEK, or TIME.

The DD statement for the ACCEPT statement is as follows:

Format

```
COBOL statement:

   ACCEPT identifier FROM mnemonic-name

JCL statement:

   //SYSIN  DD     applicable parameters
```

The *mnemonic-name* is associated with the environment name CONSOLE or the word SYSIN in the Environment Division. SYSIN is an unblocked data set that has a logical record length of 80 characters.

If you want to use a logical record length other than 80 characters, specify the length in the LRECL field of the DCB parameter, up to a maximum of 256 characters.

However, you can specify a blocked data set by using the subparameters of the DCB parameter as follows:

```
RECFM=FB,BLKSIZE=n*80
```

where *n* is the blocking factor (and SYSIN must be on a device where blocking is permitted).

Table 26 describes default FILEDEFs that are generated by COBOL/370 during application execution.

Table 26. FILEDEF Commands Issued for Running under CMS

Filename	Condition Required	Default Device	Results/Comments
SYSIN	ACCEPT FROM SYSIN	Disk	The FILEDEF is issued for: program-name SYSIN A– where the program-name is the name from the first program in the run unit's PROGRAM-ID paragraph
SYSPUNCH	DISPLAY UPON SYSPUNCH	Punch	

When the program terminates, all FILEDEFs that are created by default are automatically cleared. All other file definitions in effect are cleared when the program finishes running, except those that were specified with the PERM option.

For more information about these COBOL/370 commands, see the *IBM SAA AD/Cycle COBOL/370 Language Reference*.

Subsystem Considerations

This section describes how to run an application under CICS, DB2, and IMS.

Chapter 21. CICS Considerations 120
Mapping the CICS Program Model to the LE/370 Program Model 120
Program Execution under CICS 121
Developing an Application under CICS 121
Link-edit Considerations 121
Specifying Run-time Options under CICS 122
OS/VS COBOL Compatibility 124
Language Restrictions under CICS 124
ILC under CICS 124
Link-editing ILC applications under CICS 125
Support for Calls within Same HLL under CICS 125
Storage Management 126
CICS Short-on-Storage Condition 126
Condition Handling 126
Effect of the CICS HANDLE ABEND Command 126
Effect of CICS HANDLE CONDITION/AID 127
Restrictions on User-written Condition Handlers 127
CICS Transaction ABEND Codes 127
Using the CBLPSHPOP Run-time Option 128
User Exits 128
Run-time Output 129
Message Handling 129
Dump Services 129

Chapter 22. IMS Considerations 131
Using the Interface between LE/370 and IMS 131
IMS Communication with Your Application 132
Storage Considerations 133
Condition Handling under IMS 133
Making Your Application Reentrant 134

Chapter 23. DB2 Considerations 135
Overview of DB2 Processing 135
LE/370 Support for DB2 Applications 135
Condition Handling 135

© Copyright IBM Corp. 1991

Chapter 21. CICS Considerations

LE/370 provides support that, when used in conjunction with CICS (Customer Information Control System) facilities provided in CICS/ESA* Version 3 Release 2, permits you to write applications in High Level Languages and execute them in a CICS environment. This chapter describes special features and considerations that apply to LE/370-conforming applications running in a CICS environment.

You can code an application that runs in a CICS environment using any LE/370-conforming HLL.

There is no support for CICS on VM.

Mapping the CICS Program Model to the LE/370 Program Model

Although there is a close correlation between features of the LE/370-CICS Program Model and the LE/370-Batch Program Model described in Chapter 2, "Program Model" on page 4, some terms are defined differently.

CICS Region: A CICS Region is a fixed-size subdivision of main storage allocated to a job step or system task. For example, a CICS region is established during a CICS initialization (start-up job). The region initialization creates an environment that is common for all CICS transactions running in that environment.

CICS Transaction: A CICS "transaction" is a piece of processing initiated by a single request, usually from a terminal. A transaction consists of one or more "run-units" that carry out the needed processing when they are run.

For example, the insertion of a bank card into an ATM machine may trigger a CICS transaction consisting of one or more routines (CICS "run-units") to read the information on the card. After an ATM machine reads a bank card, the validation of the information on the card may be performed by a CICS run-unit, processing the user's personal id number may be performed by another run-unit, processing a user request another, and dispensing the cash by a final run-unit.

CICS Run-unit: A CICS run-unit consists of a statically or dynamically bound set of one or more load modules that can be loaded by the CICS program loader. A run-unit is invoked at the start of a CICS task (after a CICS transaction is triggered) or by issuing EXEC CICS LINK and EXEC CICS XCTL commands. A CICS "run-unit" is equivalent to an enclave in the LE/370-Batch Program Model.

Any link-edited load module is a run-unit in CICS; each run-unit has its own entry in the CICS Processing Program Table (PPT). Under CICS, it is possible for a single run-unit to have multiple load modules link-edited with separate entries in the PPT.

Run-units may pass control to other run-units using EXEC CICS LINK and EXEC CICS XCTL commands. The run-units are equivalent to LE/370 enclaves, and therefore each have their own heap storage and other LE/370 resources associated with an enclave.

© Copyright IBM Corp. 1991

Program Execution under CICS

The following steps describe basic application execution under CICS:

1. An event, generally the receipt of an input message containing a transaction id code, triggers a CICS "transaction".
2. CICS looks up the transaction id code in the Program Control Table (PCT) and obtains the name of the load module (or the first load module) that is to process the transaction.
3. CICS defines the transaction as an item of work dispatchable by the CICS task dispatcher.
4. Once the transaction is defined, CICS looks up the identity of the load module required to perform the task in the Processing Program Table (PPT). The PPT contains information about the load module (its language, whether it is in storage, and use count and entry point address, if so).
5. CICS calls the LE/370-CICS Run-time Level Interface to initialize the run-time environment.
6. If the load module does not perform all the processing associated with the transaction, it may invoke other load modules by a language CALL or other run-units using EXEC CICS LINK, EXEC CICS XCTL.

Developing an Application under CICS

Certain coding restrictions apply when you develop an application to run under CICS. For example, some HLL statements are not permitted under CICS; I/O operations generally must be coded using CICS commands, not HLL statements

After you code your application, you must run it through a *CICS translator*. The translator accepts as input an application containing EXEC CICS commands written in an LE/370-conforming HLL and produces as output an equivalent application where each CICS command has been translated into the language of the source. The CICS translator runs in a separate job step. The job step sequence for preparing and running an application under CICS is:

1. Code
2. Translate
3. Compile
4. Pre-link (C/370 applications only)
5. Link-edit
6. Run

For complete information on developing an application under CICS see the Programming Guide for the HLL in which your application is coded and the *CICS/ESA Application Programming Guide* listed in "Bibliography" on page 477.

Link-edit Considerations

LE/370-conforming applications that are executed under CICS can be link-edited just as if they were MVS batch applications. However, if the application uses EXEC CICS commands, you must also link-edit with your application the EXEC CICS interface stub. The name of this EXEC CICS stub is DFHELII. It is the interface routine supporting EXEC CICS commands. This stub must be available in the link-edit SYSLIB concatenation so that it can be link-edited with the application.

COBOL Considerations: DFHELII is compatible with the stub (DFHECI) that is provided for COBOL applications.

Although DFHECI is still supported under LE/370, the new interface routine provided with LE/370 offers some advantages. Whereas the old COBOL stub had to be link-edited at the top of your application, DFHELII can be linked anywhere in the application. You also have the capability of linking ILC applications with a single stub rather than with multiple stubs.

C Considerations: C applications must be link-edited AMODE(31), RMODE(ANY).

Specifying Run-time Options under CICS

Under CICS, you cannot pass run-time options as parameters when the application is invoked. However, you can specify run-time options for your application using one of the following methods:

- In the user exit (see Chapter 5, "Run-time User Exits" on page 23 for more information)
- For C/370 applications, as options specified using `#pragma runopts` (see Chapter 30, "Specifying Run-time Options" on page 206 for more information)
- As application defaults established in CEEUOPT (see Chapter 30, "Specifying Run-time Options" on page 206 for more information)
- As default options established in CEECOPT during the installation of LE/370 (see Chapter 30, "Specifying Run-time Options" on page 206 for more information).

Some run-time options have different defaults and exhibit slightly different behavior while executing under CICS. The options that differ are listed in Table 27 on page 123.

Table 27. Run-time Option Behavior under CICS

Option	Description
ABPERC	ABPERC is ignored under CICS.
AIXBLD	AIXBLD is ignored under CICS.
ALL31	ALL31(ON) is the default under CICS.
ANYHEAP	The ANYHEAP option under CICS assumes the defaults of *ANYHEAP(4K,4K,ANY,FREE).* Both the initial size and the increment size are rounded to the nearest multiple of 8 bytes. If ANYHEAP is not specified or if ANYHEAP(0) is specified, then the default value of 4K is assumed. The maximum initial and increment size for ANYHEAP is 1 gigabyte (1024M). As in batch, ANYHEAP storage is allocated on the first user request for HEAP.
ARGPARSE	ARGPARSE is ignored under CICS.
BELOWHEAP	The BELOWHEAP option under CICS assumes the defaults of *BELOWHEAP(4K,4K,FREE).* Both the initial size and the increment size are rounded to the nearest multiple of 8 bytes. If BELOWHEAP is not specified or if BELOWHEAP(0) is specified, then the default value of 4K is assumed. The maximum initial and increment size for BELOWHEAP is 65504 bytes. As in batch, BELOWHEAP storage is allocated on the first user request for HEAP.
CBLQDA	CBLQDA is ignored under CICS.
ENV	ENV is ignored under CICS.
EXECOPS	EXECOPS is ignored under CICS.
HEAP	The IBM-supplied default setting for HEAP under CICS is *HEAP(4K,4K,ANY,KEEP,4K,4K).* Both the initial size and the increment size are rounded to the next multiple of 8 bytes. If HEAP is not specified or if HEAP(0) is specified, then the default value of 4K is assumed. The maximum initial and increment size for HEAP is 1 gigabyte (1024M). As in batch, HEAP storage is allocated on the first user request for HEAP.
INTERRUPT	INTERRUPT is ignored under CICS.
LIBSTACK	The LIBSTACK option under CICS assumes the defaults *LIBSTACK(4K,4K,FREE).* Both the initial size and the increment size are rounded to the nearest multiple of 8 bytes. The maximum initial and increment size for LIBSTACK below 16M is 65504 bytes.
MSGFILE	The MSGFILE option is ignored under CICS. All messages and output (e.g., dumps, reports) are sent to a transient data queue called CESE.
PLIST	PLIST is ignored under CICS.
REDIR	REDIR is ignored under CICS.
RTEREUS	RTEREUS is ignored under CICS.
SIMVRD	SIMVRD is ignored under CICS.
STACK	The STACK option under CICS assumes the defaults of *STACK(4K,4K,ANY,KEEP).* Both the initial size and the increment size are rounded to the nearest multiple of 8 bytes. If STACK is not specified or if STACK(0) is specified, then the default value of 4K is assumed. The maximum initial and increment size for STACK below 16M will be 65504 bytes. The maximum initial and increment size for STACK above 16M is 1 gigabyte (1024M). **Note:** LE/370 uses the STACK initial size as specified in the installation defaults or programmer's defaults. LE/370 will not use the STACK initial size if the option is specified or modified in the assembler user exit or specified during debugging. Users who want to tune their run-unit execution through the STACK initial size value must relink-edit or recompile their application when a new value is desired.
STORAGE	STORAGE(NONE,NONE,NONE,0K) is the default under CICS.
TERMTHDACT	All TERMTHDACT output (including that from dumps) is written to a transient data queue named CESE.

Inheriting Run-time Options

Whenever a routine calls another routine in another run-unit using EXEC CICS LINK or EXEC CICS XCTL, the new run-unit inherits the run-time options from the first run-unit for the transaction. There is no way to set the run-time options on a run-unit by run-unit basis when EXEC CICS LINK or EXEC CICS XCTL are used to initiate the new run-unit.

Using Callable Services Under CICS

All LE/370 callable services are available to applications executing as CICS transactions. Note that for CEEMOUT (dispatch a message) and CEE3DMP (generate dump), messages and dumps are sent to a transient data queue called CESE rather than the ddname specified in MSGFILE. See "CEEMOUT — Dispatch a Message" on page 330 and "CEE3DMP — Generate Dump" on page 421 for more information.

OS/VS COBOL Compatibility

LE/370 provides a set of compatibility library routines that permit you to run OS/VS COBOL applications under CICS in compatibility mode. This compatibility library does *not* contain many of the services normally offered under LE/370. LE/370 run-time options and callable services, for example, are *not* supported.

When you run an OS/VS COBOL application on CICS, the environment that is established for a run-unit by the compatibility library routines supports only OS/VS COBOL.

Language Restrictions under CICS

In general, current language restrictions apply when LE/370-enabled applications operate under CICS. Some of the major restrictions that apply to LE/370-enabled applications operating under CICS include:

- Input/Output Restrictions - CICS provides its own I/O facilities using various EXEC CICS commands.
- Multitasking - CICS has its own multitasking capability.

For a complete description of restrictions on coding applications under CICS, see the Application Programming Guide for the language of your application listed in "Bibliography" on page 477.

ILC under CICS

LE/370 provides ILC support (with some restrictions) for applications running under CICS whose the routines are written in the HLLs listed below.

C/370 and COBOL/370: LE/370 supports interlanguage communication between routines written in IBM SAA AD/Cycle C/370 and COBOL/370 under CICS as follows:

- C/370 routines can statically call COBOL/370 routines
- COBOL/370 routines can statically call C/370 routines
- Dynamic ILC is supported in the statically called module provided that no new language is introduced into the transaction by a dynamic call. Otherwise,

dynamic calls are supported between C/370 and COBOL/370 routines as documented in "Dynamic Call/Fetch Considerations" on page 151.

- C/370 routines calling COBOL/370 routines must pass the EIB and COMMAREA as the first two parameters if the called routine has been translated.

There is *no* support for calls to or from routines written in the pre LE/370-conforming version of C/370 and COBOL.

VS COBOL II and C/370: There is no support for calls between C/370 and VS COBOL II routines.

OS/VS COBOL calling C/370: There is no support for calls between C/370 and OS/VS COBOL routines.

Assembler: There is no support for LE/370-conforming assembler main routines.

Static calls are allowed in VS COBOL II and COBOL/370 *to* but not from routines written in a pre LE/370-conforming version of assembler.

Link-editing ILC applications under CICS

You *must* link ILC applications with the new CICS stub, DFHELII, in order to obtain ILC support under LE/370. ILC applications in which C is one of the participating languages must be link-edited AMODE(31).

To statically link C/370 and COBOL routines into an ILC application, you must in some instances also include EDCSTART in the load module at link-edit time. If the main routine is C/370, this occurs automatically. If the main routine is written in COBOL, you must explicitly include EDCSTART (that is, you must specify INCLUDE SYSLIB(EDCSTART). Note that using the AUTOCALL link-edit option will *not* work; an explicit INCLUDE is necessary.

If you are migrating an old object (for example, you are running an old batch application under CICS), you can relink using the commands INCLUDE SYSLIB(EDCSTART) and REPLACE CEESTART.

For more information, see Chapter 9, "Using the Linkage-Editor Under MVS" on page 31.

Support for Calls within Same HLL under CICS

COBOL/370: Static and dynamic calls to VS COBOL II and COBOL/370 routines are supported as follows:

- Called routines can contain any command or dependency supported by CICS for COBOL.
- Calling routines must pass the EIB and COMMAREA as the first two parameters on the CALL statement, if the called routine has been translated.

There is no call support between OS/VS COBOL and COBOL/370 routines under CICS.

VS COBOL II: Static and dynamic calls to or from VS COBOL II and COBOL/370 routines are supported with the same considerations as previously listed for COBOL/370.

There is no call support between OS/VS COBOL and VS COBOL II routines under CICS.

OS/VS COBOL: OS/VS COBOL cannot call or be called by COBOL/370 or VS COBOL II routines. Communication between a routine written in OS/VS COBOL and these other versions of COBOL is permitted *only* by CICS facilities such as EXEC CICS LINK, EXEC CICS XCTL and EXEC CICS RETURN.

Storage Management

Applications may allocate and free storage explicitly through language facilities, CICS facilities by EXEC CICS GETMAIN and FREEMAIN commands (see the *CICS/ESA Application Programming Guide* for more information), or LE/370 callable services (see Chapter 34, "Dynamic Storage Callable Services" on page 267 for more information).

If you do not explicitly free storage that was allocated through language facilities or LE/370 callable services, the storage is freed at enclave (run-unit) termination.

If you issue an EXEC CICS GETMAIN (without specifying the SHARED option) to acquire storage, but do not issue an EXEC CICS FREEMAIN to free it, the storage remains allocated until the CICS transaction ends.

CICS Short-on-Storage Condition

If functions in your application attempt to acquire storage using language facilities and not enough storage is available to satisfy the request, the CICS short-on-storage condition could potentially be raised. CICS places the transaction on a stall-purgeable queue until the storage request can be satisfied.

If CICS cannot obtain enough storage to satisfy the request, and the transaction has been marked stall-purgeable in the Program Control Table then the transaction that issued the storage request is ABENDed by CICS with an ABEND code of "AKCP".

Condition Handling

LE/370 condition handling services are supported under CICS. However, there are some things to consider when using LE/370 condition handling under CICS; these are described in the following sections.

Effect of the CICS HANDLE ABEND Command

When an application is running under CICS with LE/370, condition handling will differ depending on whether:

- A CICS HANDLE ABEND is active, or
- A CICS HANDLE ABEND is *not* active.

When a CICS HANDLE ABEND is active, LE/370 condition handling will not gain control for any ABENDs or program interrupts. Any ABENDs or program interrupts

that occur while a CICS HANDLE ABEND is active cause the action defined in the CICS HANDLE ABEND to take place. The user-written condition handlers established by CEEHDLR are ignored.

When a CICS HANDLE ABEND is not active, LE/370 condition handling will gain control for ABENDs and program interrupts if the TRAP(ON) option is specified. Normal LE/370 condition handling is then performed.

Effect of CICS HANDLE CONDITION/AID

LE/370 condition handling does not modify the behavior of applications that use CICS HANDLE CONDITION or CICS HANDLE AID. The CICS CONDITION/AID conditions raised by CICS are handled only by CICS; the LE/370 condition manager is not involved in the handling of the CICS conditions.

Restrictions on User-written Condition Handlers

The following EXEC CICS commands *cannot* be used within user-written condition handlers established using CEEHDLR, or within any routine called by the user-written condition handler:

- EXEC CICS ABEND
- EXEC CICS HANDLE AID
- EXEC CICS HANDLE ABEND
- EXEC CICS HANDLE CONDITION
- EXEC CICS PUSH
- EXEC CICS POP
- EXEC CICS IGNORE CONDITION.

All other EXEC CICS commands are allowed with the user-written condition handler. However, they must be coded using the NOHANDLE option, the RESP option, or the RESP2 option. This will prevent additional conditions due to a CICS service failure.

For more information, see the applicable CICS publications listed in “Bibliography” on page 477.

COBOL considerations: A user-written condition handler registered for a routine using the CEEHDLR service cannot contain any EXEC CICS commands. This is because the CICS Translator inserts two arguments (EIB and COMMAREA) into the linkage storage section of the COBOL routine which do not match arguments passed by LE/370.

A user-written condition handler can perform EXEC CICS commands if it calls a subroutine and passes to it two dummy arguments before the real arguments. The subroutine can then perform EXEC CICS commands.

CICS Transaction ABEND Codes

The same LE/370 reserved ABEND codes (4000 through 4095) are used for applications running under CICS. In addition, there are special “reason codes” returned to CICS for LE/370 severe conditions. These severe conditions are CICS-specific. For a detailed explanation of these reason codes, see the *Language Environment/370 Debugging and Run-time Messages Guide*.

Using the CBLPSHPOP Run-time Option

This section applies to VS COBOL II and COBOL/370 routines only.

The CBLPSHPOP option controls whether the LE/370 environment automatically issues an EXEC CICS PUSH command during initialization and an EXEC CICS POP command during termination whenever a VS COBOL II or COBOL/370 subroutine is called.

As a rule of thumb, if your transaction contains calls to COBOL/370 or VS COBOL II subroutines that contain any EXEC CICS condition handling commands, specify CBLPSHPOP(ON). If your transaction contains calls to COBOL/370 or VS COBOL II subroutines that do not contain any EXEC CICS condition handling commands, specify CBLPSHPOP(OFF). You can set CBLPSHPOP on a transaction by transaction basis by using CEEUOPT.

See "CBLPSHPOP" on page 223 for more information.

User Exits

The following EXEC CICS commands *cannot* be used within the assembler user exit or any routines called by the assembler user exit:

- EXEC CICS ABEND
- EXEC CICS HANDLE AID
- EXEC CICS HANDLE ABEND
- EXEC CICS HANDLE CONDITION
- EXEC CICS PUSH
- EXEC CICS POP
- EXEC CICS IGNORE CONDITION.

All other EXEC CICS commands are allowed with the assembler user exit. However, they must be coded using the NOHANDLE option, the RESP option, or the RESP2 option. This will prevent additional conditions due to a CICS service failure.

Ensuring Transaction Rollback

Conditions that occur while an application is executing under CICS can potentially contaminate any database currently being used by the application. It is essential that a rollback is performed (that is, back out any updates made by the failing application) before further damage to the database can occur.

- When a program interrupt or ABEND is detected by CICS, the CICS operating system performs the backout automatically.
- When LE/370 detects a condition (such as a load failure), you must use an assembler user exit that transforms all abnormal terminations into operating system ABENDs in order to cause CICS to perform the rollbacks (see Chapter 27, "Advanced User Exit Topics" on page 173 for a description of the assembler user exit).

  Note: Check with your system administrator to determine the behavior of the default assembler user exit provided at your installation. An assembler user exit for CICS named CEECXITA is available in the LE/370 SCEESAMP sample library.

  Note that if TRAP(OFF) is specified and a program interrupt or an ABEND occurs, the assembler user exit is not driven.

For more information, see the applicable CICS publications listed in “Bibliography” on page 477.

Run-time Output

Message Handling

Message handling services that are provided by LE/370 under CICS, including the callable services described in Chapter 36, “Message Handling” on page 327, are the same as those under batch, with the following exceptions:

- The MSGFILE run-time option is ignored under CICS. Messages for a run-unit are directed to a CICS transient data queue named CESE.
- Messages are prefixed by a terminal ID, a transaction ID, a date, and a timestamp before their transmission. Figure 50 illustrates this format.

ASA	terminal id	transaction id	sp	Time Stamp YYYYMMDDHHMMSS	sp	Message
1	4	4	1	14	1	108

Figure 50. Format of Messages Sent to CESE

where:

ASA is the American National Standard carriage-control character.

terminal Id is a 4 character terminal identifier.

transaction Id is a 4 character transaction identifier.

sp is a space.

Time Stamp is the date and time displayed in the same format as that returned by the CEELOCT service.

Message is the message identifier and message text.

- The entire message record is preceded by an American National Standard control character to determine the format of the printing.
- Message records are V-format.

See Chapter 20, “Message Handling” on page 110 for a complete description of LE/370 message handling.

Dump Services

Dump services provided by LE/370 under CICS are the same as those under batch, with the following exception:

- The CEE3DMP callable service (see “CEE3DMP — Generate Dump” on page 421 for more information) allows you to specify in the FNAME parameter the DDNAME of the file to which the dump is written. Under CICS, the FNAME

parameter of CEE3DMP is ignored. Dumps are instead transmitted to a CICS transient data queue named CESE.

The dump format is the same as that shown in Figure 50 on page 129.

Chapter 22. IMS Considerations

LE/370 provides support to run applications under:

- IMS/VS Version 2 Release 2 and later
- IMS/ESA Version 3 Release 2.

Although you do not need to change any of the code in your application in order to run it under IMS/VS or IMS/ESA, there are a number of restrictions and recommendations that you should consider. Two of these concerns include ensuring proper condition handling under IMS and running your application under an IMS Extended Architecture environment. These topics, together with an overview of how LE/370 interacts with IMS, are discussed in detail below.

For a detailed description of how to write IMS batch and on-line applications, see the IMS Application Programming Guide listed in the "Bibliography" on page 477.

Using the Interface between LE/370 and IMS

LE/370 provides a callable service, CEETDLI, that you can use to invoke IMS facilities. Note that in pre LE/370-conforming languages, you could also invoke IMS using the following interfaces:

- in C/370 using the CTDLI interface (a `ctdli()` function call)
- in COBOL using the the CBLTDLI interface
- in Assembler using the the ASMTDLI interface.

Under LE/370, each of these interfaces continues to function in its current capacity. Although CEETDLI performs essentially the same functions, it offers some advantages, particularly if you plan to run an ILC application in IMS. For example, CEETDLI offers increased condition handling because coordination between LE/370 and IMS condition handling facilities is improved.

Only LE/370-conforming application code can call CEETDLI. Calls to CEETDLI are coded in the same way as calls to the language-specific interfaces, as follows:

Syntax

```
►►──CEETDLI──(──┬──────────────┬──function──,──┬──────┬──)──────────►◄
                └─parmcount─,──┘               └─args─┘
```

where

parmcount
: is a fullword integer specifying the total number of arguments for the CEETDLI call. This option usually isn't needed and may be omitted; it is supported for compatibility with the older interface modules.

function
: specifies the IMS function that you want to perform.

© Copyright IBM Corp. 1991

args
are arguments that you pass to IMS.

The names CEETDLI, CTDLI, CBLTDLI, and ASMTDLI are all interpreted to mean IMS interfaces. If you are currently using them in any other way in your application, you must change them.

Note that the CEETDLI interface supports calls that use either an AIB (Application Interface Block) or a PSB (Program Specification Block) parameter. For more information about AIB and a complete description of all available IMS functions and argument parameters that you can specify in CEETDLI, see the IMS Programming Guide listed in the "Bibliography" on page 477.

C/370 Considerations: To interface with IMS from C/370, you must do the following:

- Specify the `PLIST(OS)` and `ENV(IMS)` run-time options of `#pragma runopts` in your source code. The `PLIST(OS)` option establishes the correct parameter list format when invoked under IMS, and `ENV(IMS)` establishes the correct operating environment.
- When you use the `PLIST(OS)` option in `#pragma runopts`, `argc` will contain `1` (one) and `argv[0]` will contain NULL.

For more information about using the `#pragma runopts` preprocessor directive, please see Chapter 30, "Specifying Run-time Options" on page 206.

IMS/VS Compatibility Considerations

LE/370 executes under IMS/VS Version 2 Release 2 and later. However, the CEETDLI interface is *not* available under versions earlier than IMS/VS Version 3 Release 2.

IMS Communication with Your Application

When you run your application under IMS, IMS loads the application and passes to it the invocation parameter list. IMS always constructs the parameter list in the same format, regardless of the setting of the LANG= option in the PSB. A PSB is automatically scheduled for every application you run under IMS. The LANG= option has no effect on the format of the parameter list that IMS constructs. Thus, any PSB can be used with any HLL application in LE/370.

Beginning with IMV/VS Version 3 Release 2, the LANG= option in the PSB statement is not required.

Before your application is loaded, it is link-edited with an IMS language interface module, DFSLI000. Any calls that your application makes with CEETDLI for IMS services end up in this module. The IMS module in turn invokes the services and returns IMS replies to your application.

Link-edit Considerations: DFSLI000 must be link-edited with your application code. Therefore, under MVS ensure that DFSLI000 appears in a partitioned data set that is specified in the SYSLIB DD statement in your JCL used to link-edit the application. Under CMS, include DFSLI000 at link-edit time using the INCLUDE statement.

Advanced Topics

The chapters in this section describe advanced or specialized tasks that you can perform in the common run-time environment.

Chapter 26. Assembler Considerations 158
  Compatibility Considerations 158
Register Conventions 158
General Considerations 159
Using Assembler Macros 160
  CEEENTRY Macro — Generate an LE/370-conforming Prolog 160
  CEETERM Macro — Terminate an LE/370-conforming Routine 161
  CEECAA Macro — Generate a CAA Mapping 162
  CEEDSA Macro — Generate a DSA Mapping 162
  CEEPPA Macro — Generate a PPA 162
  Example of Assembler Main Routine 164
  Example of an Assembler Subroutine 165
Invoking Callable Services from Assembler Routines 170
System Services Available to Assembler Routines 170
  Using OS LINK and ATTACH Commands 172

Chapter 27. Advanced User Exit Topics 173
Using the Assembler User Exit to Tailor the Environment 173
  Using an “Installation-Wide” versus “Application-Specific” Assembler User Exit 174
  Specifying ABEND Codes to be Percolated by LE/370 175
  Actions Taken for Errors that Occur within the Assembler User Exit 175
  Assembler User Exit Interface 175
  Parameter Values in the Assembler User Exit 179
High-Level Language User Exit Interface 183

Chapter 28. Pre-initialization 185
Using the CEEPIPI Pre-initialized Interface 185
Using the PIPI Table 186
Reentrancy Considerations 188
User Exit Invocation 189
Stop Semantics 189
CEEPIPI Syntax 190
  Initialization 190
  Application Invocation 192
Service Routines 198

Chapter 29. Nested Enclaves 202
Additional Nested Enclave Considerations 202
  Run-time options 202
  Assembler User Exit 202
  Condition Handling 202
  TRAP(ON/OFF) Effects 203
  Message File 203

© Copyright IBM Corp. 1991

Chapter 26. Assembler Considerations

You can execute applications written in Assembler language in the common run-time environment. COBOL and C applications can also call or be called by assembler language applications. Regardless of whether you plan to execute a single-language assembler application or a multiple-language application containing assembler code, there are a number of restrictions you must follow under LE/370.

For example, in order to communicate with LE/370 and other applications running in the common run-time environment, the assembler application must preserve the use of certain registers and storage areas in a consistent fashion. Calling conventions for assembler programs must follow the standard S/370 linkage conventions. There are, in addition, some restrictions on the use of LE/370 storage management facilities as well as some operating system services by your assembler program under LE/370. These conventions and restrictions are described in this chapter.

Compatibility Considerations

If you are writing an LE/370-conforming assembler routine, or if your assembler routine makes any calls to LE/370 services, use the macros listed in this chapter to write LE/370-conforming assembler routines.

If you are writing a pre LE/370-conforming assembler routine, and your assembler routine does *not* make any calls to LE/370 services, follow the conventions for writing assembler routines outlined in the *VS COBOL II Application Programming Guide Release 3.1* and the *C/370 User's Guide Version 2* listed in "Bibliography" on page 477.

Assembler routines that rely on control blocks that were valid under previous versions of C and COBOL (for example, routines that check flags or switches in these control blocks) may now be invalid under LE/370. These control blocks may have changed.

CICS Considerations: LE/370-conforming Assembler is *not* supported under CICS.

Register Conventions

To communicate properly with assembler routines, you must observe certain register conventions upon entry into the assembler routine, during its execution, and upon exit from the assembler routine. These conventions are honored when you use the macros listed in this chapter to write your assembler application, or if you make a call to any LE/370 service.

Upon entry into the assembler "main" routine, these registers must contain the following values:

- R0 - address of a parameter list if the main routine is invoked from CMS
- R1 - address of the parameter list, or zero
- R13 - caller's DSA (with a valid NAB, and AMODE switching area)
- R15 - the entry point address.

© Copyright IBM Corp. 1991

Upon entry into the assembler subroutine, these registers must contain the following values:

- R0 - reserved
- R1 - address of the parameter list, or zero
- R12 - Common Anchor Area (CAA) address
- R13 - caller's DSA (with a valid NAB, and AMODE switching area)
- R14 - the return address (high order bit indicates caller's AMODE)
- R15 - the entry point address
- All other registers are undefined.

Upon entry into an assembler routine, the caller's registers (R14 - R12) are normally, but not necessarily, saved into the caller's provided DSA. After allocating a DSA (which sets the NAB field correctly in the new DSA), the first halfword of the DSA is zeroed and the backchain is set appropriately.

At all times during execution of the assembler routine, the registers contain:

- R12 - the CAA address
- R13 - the executing routine's DSA.

Upon exit from the assembler routine, the registers contain:

- R0 - undefined
- R1 - undefined
- R14 - undefined
- R15 - undefined
- All other registers are restored to their contents upon entry.

General Considerations

Condition Handling: LE/370 default condition handling actions will occur for assembler routines unless you have registered a user handler using CEEHDLR (see "CEEHDLR — Register User Condition Handler" on page 294 for more information).

Termination Processing: LE/370 relinquishes all enclave level resources that were obtained by LE/370 when the enclave terminates, and all process level services when the process terminates.

Access to the Inbound Parameter List: You can access the inbound parameter list for the assembler routine using the CEE3PRM callable service (see "CEE3PRM — Query Parameter String" on page 416). Formats of parameter lists passed to your assembler routine differ from operating system to operating system, as described in Chapter 3, "Parameter List Formats" on page 10.

Overlay Programs: LE/370 does not provide explicit support for overlay programs. However, if programs are overlay, LE/370 imposes the following restrictions:

- All LE/370 routines and static data must be placed in the root segment.
- All named routines and static data referred to by LE/370 must be in the root segment.
- All ENTRY values or static data addresses passed to any LE/370 service must point to routines in the root segment.

- All routines in the save area chain must be in storage for the whole time that they are in the chain.
- All calls must be inclusive, not exclusive (see your Linkage Editor and Loader User's Guide for the definitions of these terms).
- Calls that cause a new overlay segment to be loaded must be between two routines in the same language (that is, they may not be ILC).

Using Assembler Macros

LE/370 provides macros which generate a prolog and epilog for the assembler code. These assist in the entry and exit of assembler routines. Another macro is provided which generates the appropriate fields in the Program Prolog Area (PPA) in your assembler routine. The fields describe the entry point of an LE/370 block.

The syntax and parameter descriptions for these macros are indicated below. Included in the parameter descriptions are the IBM-supplied default values for each parameter of the macro.

CEEENTRY Macro — Generate an LE/370-conforming Prolog

CEEENTRY provides an LE/370 enabled prolog. Code is generated in co-operation with the CEEPPA macro (see "CEEPPA Macro — Generate a PPA" on page 162).

Syntax

```
►►──name──CEEENTRY──┬──────────────┬──┬──────────────┬──────────────────►
                    └─PPA=label─,──┘  └─AUTO=value─,─┘

►──┬────────────────────┬──┬─────────────────────┬──┬──────────────────────┬──►
   └─NAB=──┬─YES─┬─,────┘  └─MAIN=──┬─YES─┬─,────┘  └─PARMREG=register─,───┘
           └─NO──┘                  └─NO──┘

►──┬─────────────────┬────────────────────────────────────────────────────►◄
   └─BASE=register───┘
```

where:

name
is CSECT name of entry. This is a mandatory field.

PPA=
is the *label* of the corresponding PPA (Program Prolog Area) generated using the CEEPPA macro. If unspecified, the value of **PPA** is used.

AUTO=
is the amount of space used by prolog code for the DSA and local automatic variables that are to be allocated for the duration of this routine. This *value* must be a multiple of doublewords. If unspecified, the size of the automatic area will be the size of a DSA without any automatic variables. This is indicated by the label CEEDSASZ (this is the DSA mapping generated by the CEEDSA macro — see "CEEDSA Macro — Generate a DSA Mapping" on page 162 for more information).

NAB=

YES indicates that the previous save area has an NAB value.

NO indicates that the previous save area may not contain the NAB. Code to find the NAB is generated. This parameter is ignored for MAIN=YES.

If unspecified, **YES** is assumed.

As a rule of thumb,

- If your routine will always be called by an LE/370-conforming assembler routine, specify NAB=YES.
- If your routine may be called by a non LE/370-conforming assembler routine, specify NAB=NO.

MAIN=

YES indicates that the LE/370 environment should be brought up. This designates this assembler routine as the main routine in the enclave.

NO should be specified when the LE/370 environment is already active and only prolog code is needed. **NO** designates this assembler routine as a subroutine in the enclave.

If unspecified, **YES** is assumed.

PARMREG=

specifies the *register* to hold the inbound parameters. If unspecified, Register **1** is assumed.

BASE=

establishes the *register* that you specify here as the base register for this module. If unspecified, Register **11** is assumed.

Usage Note

1. CEEENTRY must be used in conjunction with the macros CEEDSA, CEECAA, CEETERM and CEEPPA.

CEETERM Macro — Terminate an LE/370-conforming Routine

This macro is used to terminate, or return from, an LE/370-conforming routine. If used with a main entry, the appropriate call will be made to LE/370 termination routines.

Syntax

```
▶▶──name──CEETERM──┬────────────────────┬─┬─────────────────────┬──▶◀
                   └─RC=─return_code─,─┘ └─MODIFIER=─modifier─┘
```

where:

name

is the CSECT name of entry. This is a mandatory field.

RC=

the *return code* that is to be placed into R15 after it is augmented by the **MODIFIER**, if terminating a main routine. If returning from an LE/370 subroutine, the return code itself is placed into R15, without being augmented by the **MODIFIER**.

MODIFIER=
the return code *modifier* that will be multiplied by the appropriate value (based upon the operating system) and added to the return code and placed into R15, if terminating a main routine. The MODIFIER is independently placed into R0.

CEECAA Macro — Generate a CAA Mapping

Syntax

```
▶▶──CEECAA──────────────────────────────────────────────▶◀
```

There are no parameters with this macro, nor is there a label capable of being specified. The CEECAA is required for the CEEENTRY macro.

CEEDSA Macro — Generate a DSA Mapping

Syntax

```
▶▶──CEEDSA ─────────────────────────────────────────────▶◀
```

There are no parameters with this macro. Nor is there a label capable of being specified. The minimum size of the DSA is contained in an assembler EQUATE CEEDSASZ. The CEEDSA is required for the CEEENTRY macro.

CEEPPA Macro — Generate a PPA

This macro is used to generate the LE/370 Program Prolog Area. The Program Prolog Area defines constants that describe the entry point of an LE/370 block. It is generated at assembly-time; one PPA is generated per entry point.

Syntax

```
▶▶──label─┬─────────────────────┬─┬──────────────────┬───────────▶
          └─LIBRARY=─┬─YES─┬─,─┘ └─PPA2=─┬─YES─┬─,─┘
                     └─NO──┘             └─NO──┘
▶▶─┬───────────────────────┬─┬────────────────────┬──────────────▶
   └─EXTPROC=─┬─YES─┬─,────┘ └─TSTAMP=─┬─YES─┬─,──┘
              └─NO──┘                  └─NO──┘
▶▶─┬─────────────────┬─┬───────────────────┬─┬────────────────────┬─▶
   └─PEP=─┬─YES─┬─,──┘ └─INSTOP=─┬─YES─┬─,─┘ └─EXITDSA=─┬─YES─┬─,─┘
          └─NO──┘                └─NO──┘                └─NO──┘
▶▶─┬──────────────────────┬─┬───────────────────────┬─────────────▶
   └─OWNEXM=─┬─YES─┬─,────┘ └─VER=─version_number─,─┘
             └─NO──┘
▶▶─┬───────────────────────┬─┬──────────────┬─┬─────────────────┬─▶◀
   └─REL=─release_number─,─┘ └─MOD=─level─,─┘ └─DSA=─┬─YES─┬───┘
                                                     └─NO──┘
```

where:

LIBRARY=
: indicates whether the routine is a library routine. Valid values for LIBRARY are **YES** and **NO**. If unspecified, **NO** is used. Use of this IBM-supplied default is recommended.

PPA2=
: instructs the macro to either generate a PPA2 or suppress the generation of the PPA2. Valid values for PPA2 are **YES** and **NO**. If unspecified, **YES** is used, thus generating a PPA2 field.

EXTPROC=
: indicates whether this routine is an external procedure or an internal procedure. Valid values for EXTPROC are **YES** and **NO**. If unspecified, **YES** is used thus indicating the block is an external procedure.

TSTAMP=
: indicates whether a timestamp, indicating the date and time of assembly, should be generated. Valid values for TSTAMP are **YES** and **NO**. If unspecified, **YES** is used and a timestamp is generated.

PEP=
: indicates whether this entry point is primary or secondary. Valid values for PEP are **YES** and **NO**. If unspecified, **YES** is used indicating this is a primary entry point.

INSTOP=
: indicates whether time spent in this routine should be attributed to the program (rather than the system). Valid values for INSTOP are **YES** and **NO**. If unspecified, **NO** is used indicating time should be attributed to the "system."

EXITDSA=
: indicates whether the code should gain control on GOTO out of block. Valid values for EXITDSA are **YES** and **NO**. If unspecified, **NO** is used indicating the code will not gain control for GOTO out of block. Use of this IBM-supplied default is recommended.

OWNEXM=
: specifies if this routine should participate in condition handling according to the member id set in ID. Valid values for OWNEXM are **YES** and **NO**. If unspecified, **YES** is used, indicating participation is desired. Use of this IBM-supplied default is recommended.

EPNAME=
: indicates the entry point *name*. If unspecified, the name of the CSECT will be used.

VER=
: the *version number* for the routine. This field is not interrogated by LE/370. Valid values for VER are **1** through **99**. If unspecified, **1** is used.

REL=
: the *release number* for the routine. This field is not interrogated by LE/370. Valid values for REL are **1** through **99**. If unspecified, **1** is used.

MOD=
: the modification *level* for the routine. This field is not interrogated by LE/370. Valid values for MOD are **1** through **99**. If unspecified, **0** is used.

DSA=YES
indicates whether this procedure has a DSA. Valid values for DSA are **YES** and **NO**. If unspecified, **YES** is used indicating the code does have an associated DSA. Use of this IBM-supplied default is recommended.

The CEEPPA macro is required for the CEEENTRY macro.

Example of Assembler Main Routine

Figure 59 shows a simple assembler main routine. In the example, the LE/370 environment is established, a message showing control is received in the routine, and the LE/370 environment is terminated with a non-zero return code passed in R15 to the invoker.

Note that if you write an assembler main routine, it is suggested that you nominate the routine as a load module entry point using the END statement, as shown in Figure 59. Otherwise, you must explicitly declare the routine as a main at link-edit time in any of the ways listed in Chapter 8, "Link-editing with LE/370" on page 30.

```
* ==================================================================
*
* Show a simple main assembler routine that brings up the environment
* and returns with a return code of 16, modifier of 2.
*
* ==================================================================
MAIN     CEEENTRY PPA=MAINPPA
*
*
         LA       1,PARMLIST
         L        15,=V(CEEMOUT)
         BALR     14,15
*
*   Terminate the LE/370 environment and return to the caller
*
         CEETERM  RC=16,MODIFIER=2
* ==================================================================
*              CONSTANTS AND WORKAREAS
* ==================================================================
PARMLIST DC    AL4(STRING)
         DC    AL4(DEST)
         DC    F'0'                Omitted feedback code
*
STRING   DC    AL2(STRLEN)
STRBEGIN DC    CL19'In the main routine'
STRLEN   EQU   *-STRBEGIN
DEST     DC    F'2'
MAINPPA  CEEPPA                    Constants describing the code block
         CEEDSA                    Mapping of the Dynamic Save Area
         CEECAA                    Mapping of the Common Anchor Area
         END   MAIN                Nominate MAIN as the entry point
*
```

Figure 59. Example of Simple Main Assembler Routine

Example of an Assembler Subroutine

Figure 60 illustrates a simple assembler main routine that calls the DISPARM subroutine shown in Figure 61 on page 167.

```
* ====================================================================
*
*   Show a simple main assembler routine that brings up the environment
*   and returns with a return code of 0.
*
* ====================================================================
SUBXMP   CEEENTRY PPA=XMPPPA,AUTO=WORKSIZE   MAIN = NO
         USING WORKAREA,R13
*
*
* -------------------------------------------------------------------
*
*   Setup the parameter list to CEEMOUT. Do not omit the feedback code.
*
         LA       R02,HELLOMSG
         LA       R03,DEST
         LA       R04,FBCODE
         STM      R02,R04,PLIST
*
*   Point to the parameter list and call CEEMOUT
*
         LA       R01,PLIST
         L        R15,MOUT
         BALR     R14,R15
*
*   No plist to DISPARM, so zero R1. Then call it.
*
         SR       R01,R01
         L        R15,VDISPARM
         BALR     R14,R15
*
*   Setup the parameter list to CEEMOUT. Do not omit the feedback code.
*
         LA       R02,BYEMSG
         LA       R03,DEST
         LA       R04,FBCODE
         STM      R02,R04,PLIST
*
*   Point to the parameter list and call CEEMOUT
*
         LA       R01,PLIST
         L        R15,MOUT
         BALR     R14,R15
*
*   Terminate the LE/370 environment and return to the caller
*
         CEETERM  RC=0
```

Figure 60 (Part 1 of 2). Example of an Assembler Main Routine Calling a Subroutine

```
* ==================================================================
*              CONSTANTS
* ==================================================================
MOUT     DC     V(CEEMOUT)        The LE/370 Message service
VDISPARM DC     V(DISPARM)        BAL Subroutine to display inbound parm
*
HELLOMSG DS     0F
         DC     AL2(HELLOEND-HELLOSTR)
HELLOSTR DC     C'Hello from the sub example.'
HELLOEND EQU    *
*
BYEMSG   DS     0F
         DC     AL2(BYEEND-BYESTART)
BYESTART DC     C'Terminating the sub example.'
BYEEND   EQU    *
*
*
DEST     DC     F'2'              The destination is the MSGFILE
*
XMPPPA   CEEPPA                   Constants describing the code block
* ==================================================================
*        The Workarea and DSA
* ==================================================================
WORKAREA DSECT
         ORG    *+CEEDSASZ        Leave space for the DSA fixed part
PLIST    DS     0D
PARM1    DS     A
PARM2    DS     A
PARM3    DS     A
PARM4    DS     A
PARM5    DS     A
*
FBCODE   DS     3F                Space for a 12 byte feedback code
*
*
         DS     0D
WORKSIZE EQU    *-WORKAREA
         CEEDSA                   Mapping of the Dynamic Save Area
         CEECAA                   Mapping of the Common Anchor Area
*
R00      EQU   0
R01      EQU   1
R02      EQU   2
R03      EQU   3
R04      EQU   4
R05      EQU   5
R06      EQU   6
R07      EQU   7
R08      EQU   8
R09      EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END   SUBXMP             Nominate SUBXMP as the entry point
```

Figure 60 (Part 2 of 2). Example of an Assembler Main Routine Calling a Subroutine

```
* ===================================================================
*
*   Show a simple main assembler routine that brings up the environment
*   and returns with a return code of 0.
*
* ===================================================================
DISPARM  CEEENTRY PPA=PARMPPA,AUTO=WORKSIZE,MAIN=NO
         USING WORKAREA,R13
*
*
* -------------------------------------------------------------------
*
*   Setup the parameter list to CEEPARM. Do not omit the feedback code.
*
         LA      R02,CHARPARM
         LA      R03,FBCODE
         STM     R02,R03,PLIST
*
*   Point to the parameter list and call CEEPARM
*
         LA      R01,PLIST
         L       R15,GETPARM
         BALR    R14,R15
*
*   Check the feedback code to see if everything went OK.
*
         CLC     FBCODE(8),CEE000
         BE      GOT_PARM
*
*   Setup the plist to say bad fbcode.
*
         LA      R02,BADFBC
         LA      R03,DEST
         LA      R04,FBCODE
         STM     R02,R04,PLIST
*
*   Point to the parameter list and call CEEMOUT
*
         LA      R01,PLIST
         L       R15,MOUT
         BALR    R14,R15
         B       GO_HOME               Time to go....
GOT_PARM DS      0H
*
*   See if the parm string is blank.
*
         CLC     CHARPARM(80),BLANK80     Is the parm empty?
         BNE     DISPLAY_PARM          Nope. Print it out.
*
```

Figure 61 (Part 1 of 3). Example of a Called Assembler Subroutine

```
*   Setup the plist to say empty parm.
*
         LA       R02,NOPARM
         LA       R03,DEST
         LA       R04,FBCODE
         STM      R02,R04,PLIST
*
*   Point to the parameter list and call CEEMOUT
*
         LA       R01,PLIST
         L        R15,MOUT
         BALR     R14,R15
         B        GO_HOME                Time to go....
*
DISPLAY_PARM      DS  0H
*
*   Setup the plist to CEEMOUT to display the parm.
*
         LA       R02,80                 Get the size of the string
         STH      R02,BUFFSIZE           Save it for the len-prefixed str
*
         LA       R02,BUFFSIZE
         LA       R03,DEST
         LA       R04,FBCODE
         STM      R02,R04,PLIST
*
*   Point to the parameter list and call CEEMOUT
*
         LA       R01,PLIST
         L        R15,MOUT
         BALR     R14,R15
*
*   Return to the caller
*
GO_HOME  DS       0H
         CEETERM  RC=0
* ==================================================================
*               CONSTANTS
* ==================================================================
MOUT     DC      V(CEEMOUT)        The CEL Message service
GETPARM  DC      V(CEEPARM)        The CEL Get User Parm Service
*
DEST     DC      F'2'              The destination is the MSGFILE
CEE000   DS      0D                All's well Feedback Code ...
         DC      2F'0'             ... only check the first 8 bytes of 1
BLANK80  DC      CL80' '           80 Blanks
```

Figure 61 (Part 2 of 3). Example of a Called Assembler Subroutine

```
*
BADFBC   DS    0F
         DC    AL2(BADFBEND-BADFBSTR)
BADFBSTR DC    C'Feedback code from CEEPARM was non-zero.'
BADFBEND EQU   *
*
NOPARM   DS    0F
         DC    AL2(NOPRMEND-NOPRMSTR)
NOPRMSTR DC    C'A user parm was not passed to the application.'
NOPRMEND EQU   *
*
*
PARMPPA  CEEPPA                   Constants describing the code block
* ====================================================================
*        The Workarea and DSA
* ====================================================================
WORKAREA DSECT
         ORG   *+CEEDSASZ         Leave space for the DSA fixed part
PLIST    DS    0D
PARM1    DS    A
PARM2    DS    A
PARM3    DS    A
PARM4    DS    A
PARM5    DS    A
*
FBCODE   DS    3F                 Space for a 12 byte feedback code
*
BUFFSIZE DS    H                  Halfword prefix for following string
CHARPARM DS    CL255              80 byte buffer
*
*
         DS    0D
WORKSIZE EQU   *-WORKAREA
         CEEDSA                   Mapping of the Dynamic Save Area
         CEECAA                   Mapping of the Common Anchor Area
*
R00      EQU   0
R01      EQU   1
R02      EQU   2
R03      EQU   3
R04      EQU   4
R05      EQU   5
R06      EQU   6
R07      EQU   7
R08      EQU   8
R09      EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END
```

Figure 61 (Part 3 of 3). Example of a Called Assembler Subroutine

Invoking Callable Services from Assembler Routines

The interface to a callable service is the same as the interface described above for assembler routines. An example of calling the CEEGTST ("Get Storage") callable service is shown in Figure 62.

The X'80000000' placed in the last parameter address slot indicates that the *fc* feedback code parameter is omitted.

```
*
*    R12 = A(CAA)
*    R13 = DSA
*    This example is non-reentrant.
*
         LA   R1,PLIST
         L    R15,=V(CEEGTST)
         BALR R14,R15
         . . . .
PLIST    DS   0D
         DC   A(HEAP_ID)
         DC   A(SIZE)
         DC   A(ADDR)
         DC   A(X'80000000'+FB_CODE)
HEAP_ID  DC   F'0'                      Heap-id for the user
SIZE     DC   F'256'                    Size of storage to allocate
ADDR     DC   F'0'                      Address of allocated storage
FB_CODE  DC   D                         Feedback code from the call
```

Figure 62. Sample Invocation of a Callable Service from Assembler

System Services Available to Assembler Routines

LE/370 provides a number of services that are also typically provided by the host system. Each of these services belongs to one of three categories depending upon whether it can be used in the common run-time environment:

1. The service **can** be used. You can use the host system-provided service (for example, ENQ and DEQ, for which LE/370 has no equivalent), but must manage the resource.
2. The service **should not** be used. You can use the host system-provided service, but, in doing so, it may not have the desired effect. For example, instead of using GETMAIN and FREEMAIN, it is suggested that you use the LE/370 Dynamic Storage callable services.
3. The service **must not** be used. If you use this service, it directly interferes with the LE/370 environment. For example, any ESTAE or ESPIE that you issue interferes with LE/370 condition handling.

Whenever possible, pre LE/370-conforming assembler routines should use the equivalent LE/370 services. A list of the equivalent services is provided in Table 30 on page 171.

Table 30 (Page 1 of 2). LE/370's equivalent host services		
Host service	**LE/370 Equivalent**	**Usability**
GETMAIN/FREEMAIN EXEC CICS GETMAIN/FREEMAIN	For automatic storage (block related), use LE/370's stack storage. For non-block related storage (that is, the storage persists beyond the current activation), use LE/370 heap storage.	Host services can be used, but use of equivalent LE/370 storage management services is advised.
LOAD/DELETE EXEC CICS LOAD/DELETE	No equivalent LE/370 function.	Host services can be used. The user must manage the loaded routines.
(E)STAE/(E)SPIE/SETRP/STAX EXEC CICS HANDLE ABEND	Use LE/370's condition management callable services — CEEHDLR, CEEHDLU, and CEESGL.	Host services **should not** be used. They will interfere with LE/370's condition management.
ENQ/DEQ	No equivalent LE/370 function. These may be used in the assembler routine.	These services can be used.
EXEC CICS XCTL/LINK	No equivalent LE/370 function. These may be used in the assembler routine.	These services can be used.
ATTACH/DETACH/CHAP	No equivalent LE/370 function.	These services can be used.
LINK	No equivalent LE/370 function	For compatibility, LE/370 will support the LINK boundary crossing and treat it as a new enclave.
ABEND	Call CEESGL with a severity 4 condition. Also, the assembler user exit can request an ABEND at termination. In addition, the callable service CEE3ABD can be invoked.	Host services can be used, but use of equivalent LE/370 services is advised. ABEND can be used as a last resort. However, if you issue the ABEND command, LE/370 condition handling gains control. Therefore, use of the CEE3ABD callable service is advised instead. The STEP option of ABEND should not be used due to its side effects on TSO.
WAIT/POST/EVENTS	No equivalent LE/370 function. These may be used in the assembler routine.	Host services can be used.
SNAP	Call CEE3DMP.	Host services can be used.
STIMER	No equivalent LE/370 function. This may be used in the assembler routine.	This service can be used.
TIME	Call CEE date/time services.	Host services can be used.
WTO	Call CEEMOUT. This will write to the error log or the terminal.	Host services can be used.

Table 30 (Page 2 of 2). LE/370's equivalent host services		
Host service	**LE/370 Equivalent**	**Usability**
XCTL	No equivalent LE/370 function. However, its use should be avoided.	Host services **should not** be used.
OPEN/CLOSE GET/PUT READ/WRITE	No equivalent LE/370 function.	Host services can be used.

Using OS LINK and ATTACH Commands

The COBOL STOP RUN and EXIT PROGRAM statements function differently under MVS and CMS when issued by a RES routine invoked by OS LINK or ATTACH. Under MVS, the OS LINK command creates a separate enclave; the OS ATTACH command creates a separate process and enclave. A STOP RUN in the new enclave terminates only the routine or group of routines in the new enclave created by OS LINK or ATTACH. Under CMS, OS LINK and ATTACH do *not* create a separate enclave; a STOP RUN terminates all routines in the enclave.

An EXIT PROGRAM causes a return to the routine that issued the OS LINK or ATTACH.

Chapter 27. Advanced User Exit Topics

Under LE/370, there are two user-written exit routines. One is written in assembler language and the other can be coded in any LE/370-conforming HLL *except* COBOL.

The exit routines supported under LE/370 are:

- CEEBXITA — assembler user exit
- CEEBINT — HLL user exit.

Exit routines are invoked under LE/370 to perform enclave initialization functions and both normal and abnormal termination functions. The assembler user exit, CEEBXITA, is invoked for enclave initialization, enclave termination, and process termination. The HLL user exit, CEEBINT, is invoked for enclave initialization, but not termination. The order in which these exits are driven for your application is summarized in Figure 19 (Chapter 5, “Run-time User Exits” on page 23).

You can find sample jobs containing these user exits in the SCEESAMP sample library.

The following sections describe CEEBXITA and CEEBINT as well as how to customize them to tailor the environment in which your application will run.

Using the Assembler User Exit to Tailor the Environment

The assembler user exit, CEEBXITA, allows you to tailor the characteristics of the enclave prior to its establishment. CEEBXITA must be written in Assembler language because a high-level language environment may not yet be established when the exit is invoked. It is driven for enclave initialization and enclave termination regardless of whether the enclave is the first enclave in the process or a nested enclave. The CEEBXITA exit can differentiate between the first and nested enclaves. CEEBXITA behaves differently depending upon when it is invoked, as described in the following sections.

CEEBXITA Behavior During Enclave Initialization

The CEEBXITA assembler user exit is invoked *before* enclave initialization is performed. You can use it to help guide the establishment of the environment in which your application will run. For example, you can allocate data sets in the assembler user exit. The user exit can interrogate program parameters supplied in the JCL and change them if desired. In addition, you can specify run-time options in the user exit using the **CEEAUE_OPTION** field of the assembler interface (see “Assembler User Exit Interface” on page 175 for information about how to do this).

The behavior of the IBM-supplied version of CEEBXITA differs depending upon whether you are running your application under CMS or MVS.

- Under CMS, the IBM-supplied CEEBXITA will issue FILEDEFs for CEEDUMP, SYSOUT, and SYSIN, then return control to LE/370 initialization.
- Under MVS, CEEBXITA performs no special tasks, but simply returns control to LE/370 initialization.

© Copyright IBM Corp. 1991

CEEBXITA Behavior During Enclave Termination

The CEEBXITA assembler exit is invoked *after* the user code for the enclave has completed, but *prior to* the occurrence of any enclave termination activity. For example, CEEBXITA is invoked before the storage report is produced (if one was requested), data sets are closed, and HLLs are invoked for enclave termination. In other words, the assembler user exit for termination is invoked when the environment is still active.

The CMS and MVS assembler user exits permit you to request an ABEND. Under CMS and MVS (as well as TSO and CICS under MVS), you can also request a dump to assist in problem diagnosis. Note that termination activities have not yet begun when the user exit is invoked. Thus, the majority of storage has not been modified when the dump is produced.

It is possible to request the ABEND and dump in the enclave termination user exit for all enclave-terminating events including:

- Return from the main routine
- An AD/Cycle CODE/370 QUIT command
- An HLL stop statement such as a COBOL/370 STOP RUN or C/370 `exit()`
- An unhandled condition of severity 2 or above.

CEEBXITA Behavior During Process Termination

The CEEBXITA assembler exit is invoked after:

- All enclaves have terminated
- The enclave resources have been relinquished
- Any LE/370-managed files have been closed
- AD/Cycle CODE/370 has terminated.

This allows you to de-allocate files at this time and it presents another opportunity to request an ABEND.

Using an "Installation-Wide" versus "Application-Specific" Assembler User Exit

IBM offers a default version of CEEBXITA that you have the option of using in several ways. You can simply use the IBM-supplied default version of the CEEBXITA as is. Your installation can also customize CEEBXITA for use on an installation-wide basis. When CEEBXITA is linked with the LE/370 initialization/termination library routines during installation, it functions as a "global" or installation-wide assembler user exit.

Finally, you have the ability to customize CEEBXITA yourself for use on a per-application basis. When the CEEBXITA is linked in your load module, it functions as an application-specific assembler user exit. The application-specific exit is used only when you run that application. The installation-wide assembler user exit will not be executed.

In order to obtain an application-specific user exit, you must explicitly include it at link-edit time in the application load module using an MVS INCLUDE link-edit control statement or a CMS INCLUDE command (see "Using the INCLUDE statement" on page 34 and "Issuing LOAD and INCLUDE Commands" on page 53 for more information). Any time that the application-specific exit is modified, it must be re-linked with the application.

The assembler user exit interface is described in "Assembler User Exit Interface."

Specifying ABEND Codes to be Percolated by LE/370

The assembler user exit, when invoked for initialization, may return a list of ABEND codes (on non-CICS systems, contained in the **CEEAUE_CODES** field of the assembler user exit interface - see "Assembler User Exit Interface" below) that are to be percolated by LE/370. Both system ABENDs and user ABENDs may be specified in this list.

When TRAP(ON) is in effect, and the ABEND code is in the **CEEAUE_CODES** list, LE/370 percolates the ABEND. Normal LE/370 condition handling is never invoked to handle these ABENDs. This feature is useful when you do not want LE/370 condition handling to intervene for some ABENDs; for example, when IMS issues ABEND code 777.

When TRAP(OFF) is specified, the condition handler is not invoked for any ABENDs or program interrupts.

Actions Taken for Errors that Occur within the Assembler User Exit

If any errors occur during the enclave initialization user exit, the standard system action will occur because LE/370 error handling has not yet been established.

Any errors occurring during the enclave termination user exit lead to abnormal termination (through an ABEND) of the LE/370 environment.

If there is a program check during the enclave termination user exit and TRAP(ON) is in effect, the application ends abnormally with ABEND code 4044 and Reason Code 2. If there is a program check during the enclave termination exit and TRAP(OFF) has been specified, the application ends abnormally without additional error-checking support. LE/370 performs no error handling; error handling is performed by the operating system.

LE/370 takes the same actions as described above for program checks during the process termination user exit.

Assembler User Exit Interface

You can modify CEEBXITA to perform any function desired, although the exit must have the following attributes after you modify it:

- The user-supplied exit must be named CEEBXITA.
- The exit must be reentrant.
- The exit must be capable of executing in AMODE(ANY) and RMODE(ANY).
- The exit must be re-linked with the application after modification (if you want an application-specific user exit), or re-linked with LE/370 initialization/termination routines after modification (if you want an installation-wide user exit).

If a user exit is modified, you are responsible for conforming to the interface shown in Figure 63. This user exit *must* be written in Assembler.

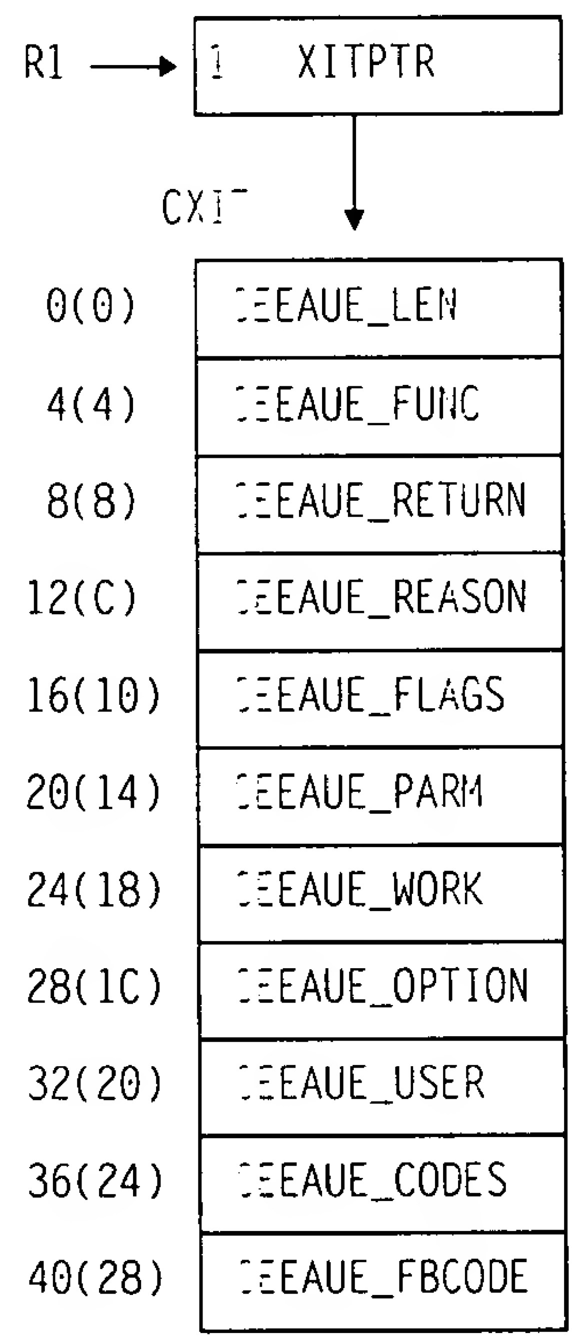


Figure 63. Interface for Assembler User Exits

When the user exit is called, Register 1 points to a word that contains the address of the CXIT control block. The high order bit is on.

The CXIT control block contains the following fullwords:

CEEAUE_LEN (input parameter)
A fullword integer that specifies the total length of this control block. For Language Environment/370 Version 1, the length is 44 bytes.

CEEAUE_FUNC (input parameter)
A fullword integer that specifies the function code. In Language Environment/370 Version 1, the following function codes are supported:

- 1 - initialization of the first enclave within a process
- 2 - termination of the first enclave within a process
- 3 - nested enclave initialization (MVS, TSO, CICS only)
- 4 - nested enclave termination (MVS, TSO, CICS only)
- 5 - process termination.

The user exit should ignore function codes other than those numbered from 1 through 5. This technique allows your user exit to operate under future LE/370 releases.

CEEAUE_RETURN (input/output parameter)
A fullword integer that specifies the return or ABEND code. CEEAUE_RETURN has different meanings depending on the function code, as follows:

- If the flag CEEAUE_ABND (see below) is off, meaning a normal return, this fullword is interpreted as the LE/370 return code that will be placed in R15.
- If the flag CEEAUE_ABND is on, meaning an ABEND has been issued, CEEAUE_RETURN is interpreted as an ABEND code that will be used when an ABEND is issued. (This could either be an EXEC CICS ABEND or an SVC 13.)

See Chapter 4, "Application Return Codes" on page 20 for more information about how LE/370 computes return/reason codes.

CEEAUE_REASON (input/output parameter)
A fullword integer that specifies the reason code for CEEAUE_RETURN.

- If the flag CEEAUE_ABND (see below) is off, this word is interpreted as the LE/370 reason code that is placed in R0.
- If the flag CEEAUE_ABND is on, CEEAUE_RETURN is interpreted as an ABEND reason code that is used when an ABEND is issued. (This field is ignored when an EXEC CICS ABEND is issued.)

See Chapter 4, "Application Return Codes" on page 20 for more information about how LE/370 computes return/reason codes.

CEEAUE_FLAGS
Contains four flag bytes. CEEBXITA uses only the first byte but reserves the remaining bytes. All unspecified bits and bytes must be zero. The layout of these flags is shown in Figure 64.

```
Byte 0
        x... .... - CEEAUE_ABTERM
        0... .... - Normal termination
        1... .... - Abnormal termination
        .x.. .... - CEEAUE_ABND
        .0.. .... - terminate with CEEAUE_RETURN
        .1.. .... - ABEND with CEEAUE_RETURN and
                      CEEAUE_REASON given
        ..x. .... - CEEAUE_DUMP
        ..0. .... - If CEEAUE_ABND=0 ABEND with no dump
        ..1. .... - If CEEAUE_ABND=1 ABEND with a dump
        ...x .... - CEEAUE_STEPS
        ...0 .... - ABEND the task
        ...1 .... - ABEND the step
        .... 0000 - Reserved bits (must be zero)
Byte 1
        00 - Reserved for future use
Byte 2
        00 - Reserved for future use
Byte 3
        00 - Reserved for future use
```

Figure 64. CEEAUE_FLAGS Format

Byte 0 (CEEAUE_FLAG1) has the following meaning:

CEEAUE_ABTERM (input parameter)
When OFF, this indicates that the enclave is terminating normally (severity 0 or 1 condition).

When ON, this indicates that the enclave is terminating with an LE/370 return code modifier of 2 or greater. This could, for example, indicate that a severity 2 or greater condition was raised that was unhandled.

CEEAUE_ABND (input/output parameter)
When OFF, this indicates that the enclave should terminate without an ABEND being issued. Thus, CEEAUE_RETURN and CEEAUE_REASON are placed into R15 and R0 and returned to the enclave creator.

When ON, terminate the enclave with an ABEND. Thus, CEEAUE_RETURN and CEEAUE_REASON are used by LE/370 in the invocation of the ABEND. While executing in CICS, an EXEC CICS ABEND command will be issued.

CEEAUE_REASON is ignored under CICS. The TRAP option does not affect the setting of CEEAUE_ABD.

CEEAUE_DUMP (output parameter)
When OFF and you request an ABEND, an ABEND will be issued without requesting a system dump.

When ON and you request an ABEND, an ABEND will be issued requesting a system dump.

Note: CMS currently honors the dump request on an ABEND if you specify the destination in one of the following FILEDEF statements:

- FILEDEF SYSABEND PRINTER
- FILEDEF SYSUDUMP PRINTER
- FILEDEF SYSMDUMP PRINTER.

CEEAUE_STEPS (output parameter)
When OFF and you request an ABEND, an ABEND will be issued to ABEND the STEP.

When ON and you request an ABEND, an ABEND will be issued to ABEND the entire TASK. This is applicable to MVS only.

Note: This is ignored under CICS and CMS.

CEEAUE_PARM (input/output parameter)
A fullword pointer to the parameter address list of the application program.

If the parameter is not a character string, CEEAUE_PARM contains the Register 1 value as passed by the calling program or operating system at the time of program entry.

If the parameter inbound to the main routine is a character string, CEEAUE_PARM contains the address of a fullword address which points to a halfword prefixed string. If this string is altered by the user exit, the string must not be extended in place.

CEEAUE_WORK (input parameter)
Contains a fullword pointer to a 256-byte work area that the exit can use. On entry it contains binary zeros and is doubleword aligned.

This area does not persist across exits.

CEEAUE_OPTION (output parameter)
Upon return, this field contains a fullword pointer to the address of a halfword length prefixed character string that contains run-time options. These options are honored during the initialization of the first enclave and any subsequent nested enclave. When invoked for enclave termination, this field is ignored.

These run-time options override all other sources of run-time options except those that are specified as non-overrideable in the installation default run-time options.

Under CICS, the STACK run time option cannot be modified using the assembler user exit.

CEEAUE_USER (input/output parameter)
Contains a fullword whose value will be maintained without alteration and passed to every user exit. Upon entry to the enclave initialization user exit, it is zero. Thereafter, the value of the user word is not altered by LE/370 or any member libraries. The user exit may change the value of this field and LE/370 maintains this value. This allows the user exit to acquire a work area, initialize it, and pass it to subsequent user exits. The work area may be freed by the termination user exit.

CEEAUE_CODES (output parameter)
During the initialization exit, this field contains a fullword address of a table of ABEND codes that the LE/370 condition handler percolates while in the (E)STAE exit. Therefore, the application is not given the opportunity to field the ABEND. The table consists of:

- A fullword count of the number of ABEND codes that are to be percolated
- A fullword for each of the particular ABEND codes that are to be percolated.

The ABEND codes may be user ABEND codes or system ABEND codes. User ABEND codes are specified by F'uuu'. For example, if you wanted user ABEND 777 to be percolated, an F'777' would be coded. System ABEND codes are specified by X'00sss000'.

This function is not enabled under CICS.

CEEAUE_FBCODE (input parameter)
Contains a fullword address of the condition token with which the enclave terminated. If the enclave terminates normally (that is, not due to a condition), the condition token is zero.

Parameter Values in the Assembler User Exit

The parameters described in "Assembler User Exit Interface" on page 175 contain different values depending on how the user exit is used. Table 31 on page 180 describes the possible values for the parameters based on how the assembler user exit is invoked.

Table 31 (Page 1 of 2). Parameter Values in the Assembler User Exit. The assembler user exit contains these parameter values depending upon when it is invoked.

When Invoked	CEEAUE_LEN	CEEAUE_RETURN	CEEAUE_REASON	CEEAUE_ FLAGS	CEEAUE_PARM
First Enclave within process initialization—Entry CEEAUE_FUNC = 1	44	0	0	0	If a character string, the R1 value pointing to a fullword, that points to a halfword prefixed string. If not a character string, the R1 value passed by the invoker.
First enclave within process initialization — Return		0, or ABEND code if CEEAUE_ABND = 1	0, or reason code for CEEAUE_RETURN if CEEAUE_ABND = 1	See Note 1 on page 182	Register 1, used as the new parameter list.
First enclave within process termination — Entry CEEAUE_FUNC = 2	44	Return code issued by application that is terminating.	Reason code that accompanies CEEAUE_RETURN.	See Note 2 on page 182	
First Enclave Within Process Termination — Return		If CEEAUE_ABND = 0 , the return code placed into R15 when the enclave terminates. If CEEAUE_ABND = 1, the ABEND code.	If CEEAUE_ABND = 0, the enclave reason code. If CEEAUE_ABND = 1, the ABEND reason code.	See Note 1 on page 182	
Nested Enclave Initialization — Entry CEEAUE_FUNC = 3	44	0	0	0	The R1 value discovered in a nested enclave creation.
Nested Enclave Initialization — Return		0, or if CEEAUE_ABND = 1, the ABEND code.	0, or if CEEAUE_ABND = 1, reason code for CEEAUE_RETURN.	See Note 1 on page 182	R1 used as the new enclave parameter list.
Nested Enclave Termination — Entry CEEAUE_FUNC = 4	44	Return code issued by enclave that is terminating.	Reason code accompanying CEEAUE_RETURN.	See Note 2 on page 182	
Nested Enclave Termination — Return		If CEEAUE_ABND = 0, the return code from the enclave. If CEEAUE_ABND = 1, the ABEND code	If CEEAUE_ABND = 0, the enclave reason code. If CEEAUE_ABND = 1, the enclave reason code.	See Note 1 on page 182	
Process Termination — Entry Function Code = 5	44	Return code presented to the invoking system in R15 that reflects the value returned from the "first enclave within process termination."	Reason code accompanying CEEAUE_RETURN that is presented to the invoking system in R0 and reflects the value returned from the "first enclave within process termination."	See Note 3 on page 182	
Process Termination — Return		If CEEAUE_ABND = 0, return code from the process. If CEEAUE_ABND = 1, the ABEND code.	If CEEAUE_ABND = 0, the reason code for CEEAUE_RETURN from the process. If CEEAUE_ABND = 1, reason code for the CEEAUE_RETURN ABEND reason code.	See Note 1 on page 182	

Table 31 (Page 2 of 2). Parameter Values in the Assembler User Exit. The assembler user exit contains these parameter values depending upon when it is invoked.

When Invoked	CEEAUE_WORK	CEEAUE_OPTION	CEEAUE_USER	CEEAUE_ CODES	CEEAUE_ FBCODE
First Enclave Within Process Initialization — Entry CEEAUE_FUNC = 1	Address of a 256-byte work area of binary zeros.		0		
First Enclave within Process initialization — Return		Pointer to address of a halfword prefixed character string containing run-time options, or 0.	Value of CEEAUE_USER for all subsequent exits.	Pointer to the ABEND codes table, or 0.	
First enclave within process Termination — Entry CEEAUE_FUNC = 2	Address of a 256-byte area of binary zeros.		Return value from previous exit.		Feedback code causing termination.
First Enclave Within Process Termination — Return			The value of CEEAUE_USER for all subsequent exits.		
Nested Enclave Initialization — Entry CEEAUE_FUNC = 3	Address of a 256-byte work area of binary zeros.		Return value from previous exit.		
Nested Enclave Initialization — Return		Pointer to fullword address that points to a halfword prefixed length string containing run-time options, or 0	The value of CEEAUE_USER for all subsequent exits.	Pointer to ABEND Codes table, or 0.	
Nested Enclave Termination — Entry CEEAUE_FUNC = 4	Address of a 256-byte work area of binary zeros		Return value from previous exit.		Feedback code causing termination.
Nested Enclave Termination — Return			Value of CEEAUE_USER for all subsequent exits.		
Process Termination — Entry CEEAUE_FUNC = 5	Address of a 256-byte work area of binary zeros.		Return value from previous exit.		Feedback code causing termination.
Process Termination — Return			Value of CEEAUE_USER for all subsequent exits.		

Notes:

1. **CEEAUE_FLAGS:**
   CEEAUE_ABND = 1 if an ABEND is requested, or 0 if the enclave should continue with execution
   CEEAUE_DUMP = 1 if the ABEND should request a DUMP
   CEEAUE_STEPS = 1 if the ABEND should ABEND the step
   CEEAUE_STEPS = 0 if the ABEND should ABEND the task.
2. **CEEAUE_FLAGS:**
   CEEAUE_ABTERM = 1 if the application is terminating with a LE/370 return code modifier of 2 or greater, 0 otherwise.
3. **CEEAUE_FLAGS:**
   CEEAUE_ABTERM = 1 if the last enclave is terminating abnormally (that is, an LE/370 return code modifier is 2 or greater). This reflects the value returned from the "first enclave within process termination"

High-Level Language User Exit Interface

Language Environment/370 Version 1 defines an HLL user exit for enclave initialization that you can code in C/370 or LE/370-conforming assembler. Note that the HLL User Exit *cannot* be written in COBOL. COBOL programmers may use an HLL exit written in C/370, one written in LE/370-conforming assembler or default to the IBM-supplied default HLL user exit (which is written in C/370).

The HLL enclave initialization exit is invoked after the enclave has been established, after the AD/Cycle CODE/370 initial command string has been processed, and prior to the invocation of compiled code. The name of this exit is CEEBINT. When invoked, it is passed a parameter list that conforms to the LE/370 definition. The parameters are all fullwords and are defined as follows:

Number of arguments in parameter list (input)
a fullword binary integer.

- Upon entry: Contains 7
- Upon exit : Not applicable

Return code (output)
a fullword binary integer.

- Upon entry: 0
- Upon exit : Able to be set by the exit, but not interrogated by LE/370.

Reason code (output)
a fullword binary integer.

- Upon entry: 0
- Upon exit : Able to be set by the exit, but not interrogated by LE/370.

Function code (input)
a fullword binary integer.

- Upon entry: 1, indicating the exit is being driven for initialization.
- Upon exit : Not applicable.

Address of the main program entry point (input)
a fullword binary address.

- Upon entry: The address of the routine that gains control first.
- Upon exit : Not applicable.

User word (input/output)
a fullword binary integer.

- Upon entry: Value of the user word (CEEAUE_USER) as set by the assembler user exit.
- Upon exit : The value set by the user exit, maintained by LE/370 and passed to subsequent user exits.

Exit List Address (output)
a fullword binary integer reserved for future use.

This allows the establishment of one or more user exits when the enclave user

exit sets this field to a list of user exits. Currently, only one user exit is supported in Language Environment/370 Version 1.

Usage Notes

1. The user exit must not be a main-designated routine. That is, it must not be a C/370 main function.
2. The HLL exit routines must be linked with compiled code. If you do not provide an initialization user exit, an IBM-supplied default, which simply returns control to your application, is linked with the compiled code.
3. The exit cannot be written in COBOL/370.
4. The exit should be coded so that it returns for all unknown function codes.
5. HLL constructs such as the C/370 `exit()`, `abort()`, `raise(SIGTERM)`, and `raise(SIGABRT)` functions terminate the enclave.

Chapter 28. Pre-initialization

Pre-initialization allows an application to initialize the HLL environment once, perform multiple executions using the environment, and then explicitly terminate the environment. From a non LE/370-conforming driver (such as Assembler) you can use LE/370 pre-initialization facilities to create and initialize a common run-time environment, execute applications written in an LE/370-conforming HLL multiple times within the pre-initialized environment, and terminate the pre-initialized environment.

LE/370 pre-initialization offers enhanced performance for repeated invocations of your application. For example, if an assembler routine invokes either a number of LE/370-conforming HLL routines or the same HLL routine a number of times, the creation and termination of that HLL environment multiple times is unnecessarily inefficient. The solution is to create the HLL environment only once for use by all invocations of the routine. This can be achieved by pre-initializing the environment.

In the pre-intialized environment, the first routine to execute can be treated as either the main routine or a subroutine of that execution instance. LE/370 provides support for both of these types of pre-initialized routines:

- Executing main routine multiple times
- Executing subroutines multiple times.

C/370 Compatibility Concerns: LE/370 honors the current C/370 assembler interface to pre-initialization by calls to CEESTART with a special extended parameter list. For more information about this interface, see the *IBM SAA AD/Cycle C/370 Programming Guide.*

COBOL/370 Compatibility Concerns: LE/370 honors the current COBOL/370 interfaces to pre-initialization, RTEREUS, ILBOSTP0, and IGZERRE. For more information about these interfaces, see the *IBM SAA AD/Cycle COBOL/370 Migration Guide*.

Using the CEEPIPI Pre-initialized Interface

The interface for pre-initialized routines is a loadable routine called CEEPIPI. It is loaded as an RMODE(24)/AMODE(ANY) routine and returns in the AMODE of its caller when the request is satisfied.

CEEPIPI handles the requests for environment initialization, application invocation, and environment termination. All requests for services by CEEPIPI must be made from a non-LE/370 environment. The parameter list for CEEPIPI is an OS standard linkage parameter list. Each request to CEEPIPI is identified by a function code, the first parameter in the parameter list. The function code is a fullword integer (for example, 1 = init_main, 2 = call_main).

The pre-initialization services offered under LE/370 are listed in Table 32.

Table 32. Pre-initialization Services Accessed Using CEEPIPI

Function code	Integer value	Service performed
init_main	1	Create and initialize an environment for multiple executions of main routines
call_main	2	Invoke a main routine within an already initialized environment
init_sub	3	Create and initialize an environment for multiple executions of subroutines
call_sub	4	Invoke a subroutine within an already initialized environment
term	5	Explicitly terminate the environment without executing a user routine
add_entry	6	Dynamically add a candidate routine to execute within the pre-initialized environment

The section "CEEPIPI Syntax" on page 190 contains a detailed description and information about how to invoke each of these services.

VM Considerations: Depending on how LE/370 is installed under VM, CEEPIPI can reside in an NSS, reside as a member in a load library, or could be a relocatable load module. You must know this in order to issue the appropriate SVC commands to load and delete CEEPIPI. Check with your system administrator to learn how CEEPIPI is installed at your location.

Using the PIPI Table

The "PIPI table" is used by LE/370 to identify the routines that are candidates for execution in the pre-initialized environment, as well as to optionally load the routine when it is called. It is possible to have an empty PIPI table with no entries. The PIPI table contains the names and the entry point addresses of each routine that may be executed within the pre-initialized environment. Candidate routines may be present in the table when the init_main or init_sub functions are invoked, or may be added to the table using CEEPIPI(add_entry).

COBOL considerations: The *only* COBOL routines that can be the target of CEEPIPI(call_main) or CEEPIPI(call_sub) are COBOL/370 routines (not VS COBOL II or OS/VS COBOL routines).

C considerations: The only C routines that can be the target of CEEPIPI(call_main) or CEEPIPI(call_sub) are IBM SAA AD/Cycle C/370 routines. Routines written in the pre-LE/370 conforming version of C/370 (Version 2 Release 1) can be included in the table only if you relink the old load module with LE/370's CEESTART.

Using Macros to Generate the PIPI Table

LE/370 provides the following Assembler macros to generate the PIPI table for you:

- **CEEXPIT** - generates a header for the PIPI table. This macro has no parameters.

Syntax

```
▸▸——table_name——CEEXPIT——————————————————————————————▸◂
```

table_name
: an assembler symbolic name assigned to the first word in the PIPI table. This is the value that should be used as the *ceexptbl_addr* parameter in a CEEPIPI(init_main) or a CEEPIPI(init_sub) call.

- **CEEXPITY** - generates an entry within the PIPI table

Syntax

```
▸▸——CEEXPITY—(——————————,————————————————)——————————————▸◂
              └─name─┘   └─entry_point─┘
```

name
: the first eight characters of the load name of a routine that can be invoked within the LE/370 pre-initialized environment.

entry_point
: the address of the load module that is to be invoked.

You have the option of specifying either, both, or neither of the parameters:

- If *name* is omitted and *entry_point* is present, the comma must be present.
- If both parameters are omitted, the entry is a candidate for assignment to the PIPI table by a call to CEEPIPI(add_entry).
- If both parameters are present, *name* is ignored and *entry_point* is used as the start of the routine.

Each invocation of the CEEXPITY macro generates a row in the PIPI table. The first entry is row 0, the second is row 1, and so on.

When your routine is invoked, LE/370 automatically determines its AMODE. Control is returned to the caller of your routine in the caller's AMODE.

- **CEEXPITS** - identifies the end of the PIPI table.

This macro has no parameters.

Syntax

```
▸▸——CEEXPITS—————————————————————————————————————————▸◂
```

Reentrancy Considerations

You can make multiple calls to main routines by invoking CEEPIPI with a CALL request multiple times using a single PIPI table. However several conditions apply, including whether the called routines are 'MAIN' routines and are reentrant.

If your routine is:

- A main routine invoked using CEEPIPI(call_main)
- Reentrant
- Using external variables,

then when your routine is invoked again, the external variables are re-initialized. Multiple executions of a reentrant main routine are not influenced by a previous execution of the same routine.

On the other hand, if your routine:

- Is a main routine invoked using CEEPIPI(call_main)
- Is *not* reentrant
- Uses external variables,

then when your routine is invoked again, the external variables can potentially contain last-used values. Local variables (those contained in the object code itself) may also contain last-used values. If main routines are allowed to execute multiple times, a given execution of a routine can influence subsequent executions of the same routine.

Rules of thumb to observe are:

- Specify only reentrant routines for multiple invocations.
- If your routine is not reentrant, you may see unintended results.

User Exit Invocation

User exits are invoked for initialization and termination during calls to CEEPIPI as shown in Table 33.

Table 33. Invocation of User Exits during Process and Enclave Initialization and Termination

Function	When Invoked
Assembler user exit for first enclave initialization	• CEEPIPI(init_sub) • CEEPIPI(call_main) • CEEPIPI(call_sub) if a previous CEEPIPI(call_sub) ended with stop semantics (see "Stop Semantics")
HLL user exit	• CEEPIPI(init_sub) • CEEPIPI(call_main) • CEEPIPI(call_sub) if a previous CEEPIPI(call_sub) ended with stop semantics
Assembler user exit for first enclave termination	• CEEPIPI(call_main) • CEEPIPI(call_sub) which issues a STOP in COBOL or an `exit()` in C/370 • CEEPIPI(term) for environment created with CEEPIPI(call_sub) if the last CEEPIPI(call_sub) did not end with stop semantics
C/370 atexit functions	• CEEPIPI(call_main) • CEEPIPI(call_sub) which issues a STOP in COBOL or an `exit()` in C/370 • CEEPIPI(term) for environment created with CEEPIPI(call_sub) if the last CEEPIPI(call_sub) did not end with stop semantics
Assembler user exit for process termination	• CEEPIPI(term)

See Chapter 27, "Advanced User Exit Topics" on page 173 for more information about user exits.

Stop Semantics

When the one of the following is issued within the pre-initialized environment for subroutines:

- COBOL/370 STOP or STOP RUN statement;
- C/370 `exit()`, `return()`, or `abort()`.

or when an unhandled condition will cause termination of the (only) thread, the logical enclave will be terminated. The *process* level of the environment will be retained. LE/370 will **not** delete those entries that were loaded explicitly by LE/370 during the pre-initialization processing.

CEEPIPI Syntax

The following section describes how to invoke the CEEPIPI interface to perform the following tasks:

- Initialization
- Application invocation
- Termination
- Adding an entry to the PIPI table.

Initialization

An LE/370 environment can be initialized in two different capacities - one to allow executions of *main* routines, the other to allow multiple executions of subroutines. Each capacity is discussed below.

CEEPIPI(init_main) —Initialize for Main Routines

This invocation of CEEPIPI:

- Creates and initializes a new common run-time environment (process and enclave) that allows the execution of main routines multiple times
- Sets the environment dormant so that exceptions are percolated out of it
- Returns a token identifying the environment to the caller.

Syntax

```
▸▸——CALL——CEEPIPI—(—init_main—,—ceexptbl_addr—,——————————▸

▸——service_rtns—,—token——)——————————————————————————————▸◂
```

init_main **(input)**
: a fullword function_code (integer value = 1) containing the init_main request.

ceexptbl_addr **(input)**
: a fullword containing the address of the PIPI Table to be used during initialization of the new environment. LE/370 does not alter the user-supplied copy of the table. If an entry address is zero and the entry name is nonblank, LE/370 searches for the routine (in the LPA, NSS, nucleus — see Chapter 6, "LE/370 Library Routine Considerations" on page 27 for a detailed description of locations where dynamic routines are stored under MVS and CMS) and dynamically loads it. LE/370 places the entry address in the corresponding slot of an LE/370-maintained table.

service_rtns **(input)**
: a fullword containing the address of the service routine vector or 0, if there is no service routine vector. See "Service Routines" on page 198 for more information.

token **(output)**
: a fullword containing a unique value used to represent the environment

  Token should be used only as input to additional calls to CEEPIPI, and should not be altered or used in any other manner.

R15 contains a return code indicating whether an environment was successfully initialized or not. The return codes are as follows:

0	A new environment was successfully initialized.
4	The function_code is invalid.
8	All addresses in the table were not resolved. This can occur if a LOAD failure was encountered or a routine within the table was not generated by an LE/370-conforming HLL.
16	CEEPIPI was called from an active environment.

CEEPIPI(init_sub) — Initialize for Subroutines

This invocation of CEEPIPI:

- Creates and initializes a new common run-time environment (enclave) that allows the execution of subroutines multiple times
- Sets the environment dormant so that exceptions are percolated out of it
- Returns a token identifying the environment to the caller.

Syntax

```
▸▸——CALL——CEEPIPI—(—init_sub—,—ceexptbl_addr—,——————————▸

▸——service_rtns—,—runtime_opts—,—token—)————————————————▸◂
```

init_sub **(input)**
: a fullword function_code (integer value = 3) containing the init_sub request.

ceexptbl_addr **(input)**
: a fullword containing the address of the PIPI Table to be used during initialization of the new environment. LE/370 does not alter the user-supplied copy of the table. If an entry address is zero and the entry point is non-blank, LE/370 searches for the routine (in the LPA, NSS, nucleus — see Chapter 6, "LE/370 Library Routine Considerations" on page 27 for a detailed description of locations where dynamic routines are stored under MVS and CMS) and dynamically loads it. LE/370 then places the entry address in the corresponding slot of an LE/370-maintained table.

service_rtns **(input)**
: a fullword containing the address of the service routine vector or 0, if there is no service routine vector. See "Service Routines" on page 198 for more information.

run-time_opts **(input)**
: a fixed-length 255-character string containing run-time options (see Chapter 31, "Run-time Options" on page 213 for a list of run-time options that you can specify).

token **(output)**
: a fullword containing a unique value used to represent the environment.

  Token should be used only as input to additional calls to CEEPIPI, and should not be altered or used in any other manner.

R15 contains a return code indicating the success or failure of the call. The return codes are as follows:

0	A new environment was successfully initialized.
4	The function code is invalid.
8	All addresses in the table were not resolved. This can occur if a LOAD failure was encountered or a routine within the table was not generated by an LE/370-conforming HLL.
16	CEEPIPI was called from an active environment.

Application Invocation

LE/370 provides two facilities to invoke either a main or subroutine. When invoking main routines, the environment must have been initialized using the init_main function code. In a similar manner, when invoking subroutines, the environment must have been initialized with the init_sub function code.

CEEPIPI(call_main) —Invocation for Main Routine

This invocation of CEEPIPI invokes as a main routine the routine that you specify. The common execution environment identified by the *token* is activated before the called routine is invoked, and after the called routine returns, the environment is dormant.

Syntax

```
▸▸——CALL——CEEPIPI—(—call_main—,—ceexptl_index—,————————————▸
▸——token—,—runtime_opts—,—parm_ptr—,—appl_return_code—,————▸
▸——appl_reason_code—,—appl_feedback_code—)—————————————————▸◂
```

call_main **(input)**
: a fullword function_code (integer value = 2) containing the call_main request.

ceexptbl_index **(input)**
: a fullword containing the row number within the PIPI table of the entry that should be invoked. The index starts at 0.

Note that each invocation of the CEEXPITY macro generates a row in the PIPI table. The first entry is row 0, the second is row 1 and so on. A call to CEPIPI(add_entry) to add an entry to the PIPI table will also return a row number in the *ceexptbl_index* parameter.

token **(input)**
: a fullword with the value of the token returned by CEEPIPI(init_main) when the common run-time environment is initialized.

runtime_opts **(input)**
: a fixed-length 255-character string containing run-time options (see Chapter 31, "Run-time Options" on page 213 for a list of run-time options that you can specify).

parm_ptr **(input)**
a fullword parameter list pointer or 0 (zero) that is placed into R1 when the main routine is executed. Run-time options are **not** obtained from this parameter.

The parameter list that is passed must be in a format that HLL subroutines will expect (for example, in an `argc, argv` format for C/370 routines).

appl_return_code **(output)**
a fullword containing the return code returned by the called routine when it finished executing.

If the called routine returns no return code, *appl_return_code* is 0.

appl_reason_code **(output)**
a fullword containing the reason code returned by the environment when the routine finished executing.

If the called routine returns no reason code, *appl_reason_code* is 0.

appl_feedback_code **(output)**
a 96-bit condition token indicating why the application terminated.

A return code is provided in register 15 and may contain the following values:

0	The environment was activated and the routine called.
4	The function_code is invalid.
8	CEEPIPI was called from an LE/370-conforming HLL.
12	The indicated environment was initialized for subroutines. No routine was executed.
16	The *token* is invalid.
20	The index points to an entry that is invalid or empty.
24	The index that was passed is outside the range of the table.

Usage Notes

1. The *token* must identify a previously pre-initialized environment that is not active at the time of the call.
2. The user return code and LE/370 reason code are set to zero prior to invoking the target routine.
3. At termination, the HLL event handlers that are currently active are driven to enforce language semantics for the termination of an application such as closing files and freeing storage.
4. If the routine requests normal termination, for example by a COBOL STOP RUN, the routine terminates and returns to the caller of the pre-init call service, but the process level is made dormant rather than terminated. The thread and enclave levels are terminated. The assembler user exit is driven with the function code for first enclave termination (see Chapter 27, "Advanced User Exit Topics" on page 173 for more information about user exits).
5. If the routine terminates abnormally (ABENDs) or a severity 4 condition is encountered, the environment is terminated. The return code indicates that this action has occurred.

6. The *appl_return_code* value returned is the R15 value that is usually returned by the application at the end of the enclave execution. That is, it is the return code that would be presented to the host system.
7. The *appl_reason_code* value returned is the R0 value that is usually returned by the application at the end of the enclave execution. That is, it is the reason code that would be presented to the host system.
8. The NOEXECOPS and CBLOPTS run-time options are ignored since the parameter inbound to the application and the run-time options are separated already. Therefore, NOEXECOPS and CBLOPTS do not affect the parameter string format.

CEEPIPI(call_sub) — Invocation for Subroutines

This invocation of CEEPIPI invokes as a subroutine the routine that you specify. The common run-time environment identified by the *token* is activated before the called routine is invoked, and after the called routine returns, the environment is dormant.

The enclave is terminated when an unhandled condition is encountered or a STOP statement is executed (see "Stop Semantics" on page 189 for more information). However, the process level will be maintained.

Syntax

```
►►——CALL——CEEPIPI—(—call_sub—,—ceexptl_index—,—token—,——————►
►——parm_ptr—,—sub_ret_code—,—sub_reason_code—,————————————————►
►——sub_feedback_code—)————————————————————————————————————————►◄
```

call_sub **(input)**
: a fullword function_code (integer value = 4) containing the call_sub request for a subroutine.

ceexptbl_index **(input)**
: a fullword containing the row number of the entry within the PIPI table that should be invoked. The index starts at 0.

token **(input)**
: a fullword with the value of the token returned when the common run-time environment is initialized. This token is returned by the CEEPIPI(init_sub) request during the initialization call.

parm_ptr **(input)**
: a parameter list pointer or 0 (zero) that is placed into R1 when the routine is executed. Run-time options are **not** obtained from this parameter.

  C/370 users are advised to follow the subroutine linkage convention for C/370—Assembler ILC apllications as outlined in the IBM SAA AD/Cycle C/370 Programming Guide.

sub_ret_code **(output)**
: the subroutine return code.

  If the enclave is terminated due to an unhandled condition or a STOP, this contains the enclave return code for termination.

If the called subroutine returns no return code, *sub_return_code* is 0.

sub_reason_code **(output)**
the subroutine reason code. This is 0 for normal subroutine returns. If the enclave is terminated due to an unhandled condition or a STOP, this contains the enclave reason code for termination.

sub_feedback_code **(output)**
the feedback code for enclave termination. This is the **CEE000** feedback code for normal subroutine returns. If the enclave is terminated due to an unhandled condition or a STOP, this contains the enclave feedback code for termination.

A return code is provided in Register 15 and can contain the following values:

0	The environment was activated and the routine called.
4	The function_code is invalid.
8	CEEPIPI was called from an LE/370-conforming HLL.
12	The indicated environment was initialized for main routines. No routine was executed.
16	The *token* is invalid.
20	The index points to an entry that is invalid or empty.
24	The index passed is outside the range of the table.
28	The enclave was terminated but the process level persists.

Usage Notes

1. If the subroutine issues a STOP statement or an unhandled condition arises that causes the enclave to terminate, the enclave is terminated. However, the process level is not terminated. When the enclave level is terminated, any subsequent invocation creates a new enclave using the same run-time options used in the creation of the first enclave. LE/370 does not delete any user routines that were loaded into the PIPI table.

   It is your responsibility to ensure that external data is in its initial state.

2. Return code 28 indicates the enclave was terminated while the process was retained. This can occur due to a STOP statement being issued or due to an unhandled condition.

3. The *token* must identify a previously pre-initialized environment that is not active at the time of the call. You must not alter the value of the token.

4. If the routine terminates abnormally (ABEND) or a severity 4 condition is encountered, the enclave is terminated and the *sub_ret_code, sub_reason_code,* and *sub_feedback_code* indicates this action.

CEEPIPI(term) — Terminate Environment

This invocation of CEEPIPI terminates the environment identified by the value given in *token*. This service is used for terminating environments created for subroutines or main routines.

Syntax

```
▸▸——CALL——CEEPIPI—(—term—,—token—,—env_return_code—)——▸◂
```

term **(input)**
: a fullword function_code (integer value = 5) containing the termination request.

token **(input)**
: a fullword with the value of the token of the environment to be terminated. This token is returned by CEEPIPI(init_main) or CEEPIPI(init_sub) request during the initialization call.

env_return_code **(output)**
: a fullword integer which is set to the return code from the environment termination.

  If the environment was initialized for a main or a subroutine, and the last CEEPIPI(call_sub) issued stop semantics, the value of *env_return_code* will be zero.

  If the environment was initialized for a subroutine, and the last CEEPIPI(call_sub) did not terminate with stop semantics, *env_return_code* will contain the same value as that in *sub_ret_code* from the last CEEPIPI(call_sub).

Upon return, R15 contains a return code indicating the success or failure of this request and can contain the following values:

0	The environment was activated and termination was requested.
4	Invalid function code.
8	CEEPIPI was called from an LE/370-conforming routine.
16	The *token* is invalid.

Usage Notes

1. *Token* must identify a previously pre-initialized environment that is dormant at the time of the call.
2. All resources obtained are released when the environment terminates.
3. All routines loaded by LE/370 are deleted when the environment terminates.
4. Subsequent references to *token* by pre-init services results in an error indicating the token is invalid.

CEEPIPI(add_entry) — Add an Entry to the PIPI Table

This invocation of CEEPIPI adds an entry for the environment represented by the *token* into the LE/370-maintained table. If a routine entry address is not provided, the routine name is used to dynamically load the routine and add it to the PIPI table. The PIPI table index for the new entry is returned to the calling routine.

Syntax

```
▸▸——CALL——CEEPIPI—(—add_entry—,—token—,—routine_name—,——▸
▸——routine_entry—,—ceexptbl_index—)——————————————▸◂
```

add_entry **(input)**
a fullword function_code (integer value = 6) containing the add_entry request.

token **(input)**
a fullword with the value of the token associated with the environment that will add this new routine. This token is returned by a CEEPIPI(init_main) or CEEPIPI(init_sub) request.

routine_name **(input)**
a character string of length 8, left-justified and padded right with blanks, containing the name of the routine. To indicate the absence of the name, this field should be blank. If *routine_entry* is zero, this is used as the load name.

routine_entry **(input/output)**
the routine entry address that is added to the PIPI table. If routine_entry is zero on input, then *routine_name* is used as the load name. On output, routine_entry is set to the load address of routine_name.

ceexptbl_index **(output)**
the index into the PIPI table where this routine was added. If the return code is non-zero, this value is indeterminate. The index starts at zero.

Upon return, R15 contains a return code indicating the success or failure of this request and may contain the following values:

0	The routine was added to the PIPI table.
4	Invalid function code.
8	CEEPIPI was called from an LE/370-conforming routine.
12	The routine did not contain a common run-time environment PPA style prolog. The PIPI table was not updated. *routine_entry* is set to the address of the loaded routine.
16	The *token* is invalid.
20	The routine name contains only blanks and the routine entry address was zero. The PIPI table was not updated.
24	The *routine_name* was not found or there was a load failure. The PIPI table was not updated.
28	The PIPI table is full. No routine was added to the table, nor was any routine loaded by LE/370.

Usage Notes

1. The PIPI table is built using the macros described in this chapter. Therefore, its size is under the control of your application, not LE/370.
2. *token* must identify a previously pre-initialized environment that is dormant at the time of the call.

3. None of the routines in the PIPI table can be nested routines. All routines must be external routines.

Service Routines

Under LE/370, you can use specify several service routines to execute a main or subroutine in the pre-initialized environment. To use the routines, specify a list of addresses of the routines in a service routine vector as shown in Figure 65.

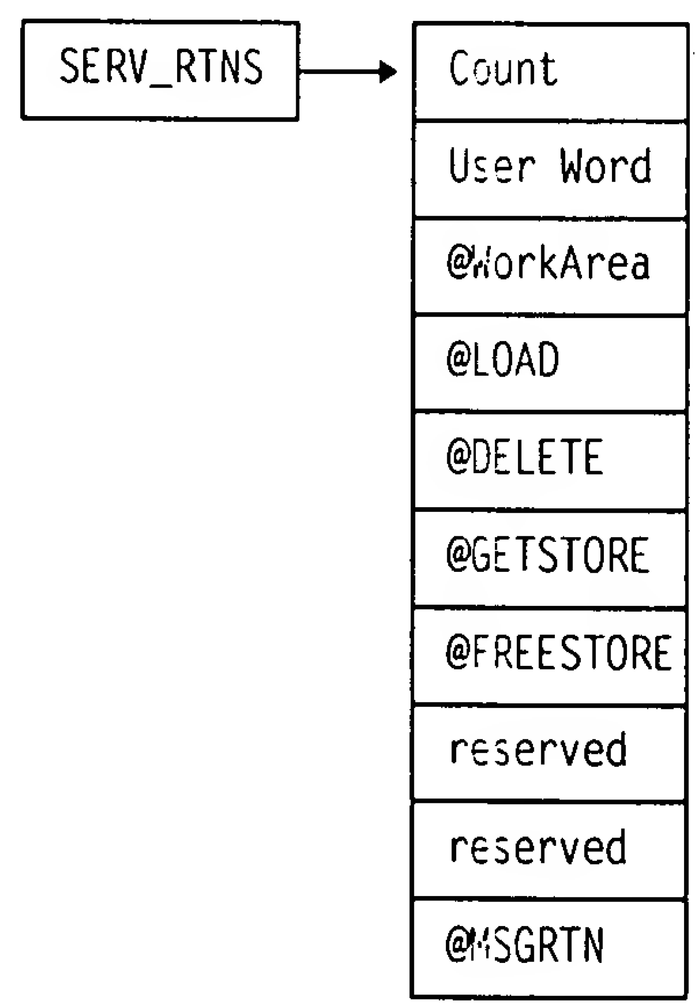


Figure 65. Format of Service Routine Vector

The service routine vector is comprised of a list of fullword addresses of routines that will be used *instead* of LE/370 service routines. The list of addresses is preceded by the number of the addresses in the list, as specified in the *count* field of the vector. The *service_rtns* parameter that you specify in calls to CEEPIPI(init_main) and CEEPIPI(init_sub) contains the address of the vector itself. If this pointer is specified as zero (0), then LE/370 routines are used instead of the service routines shown in Figure 65.

Note that the addresses of the @GETSTORE, @FREESTORE, @LOAD, and @DELETE service routines must be presented together in the service routine vector. If any one of these addresses is zero, none of the services are used.

The service routines must be AMODE(ANY) / RMODE(24).

Count
: A fullword binary number representing the number of fullwords that follow. The *count* does not include itself. In Figure 65, the count is 9. For each vector slot, a zero represents the absence of the routine, a non-zero represents the presence of a routine.

User Word
: A fullword that is passed to the service routines. The *user word* is provided as a means for your routine to communicate to the service routines.

@WorkArea
An address of a workarea of at least 256 bytes that is doubleword aligned. The first word of the area contains the length of the area provided.

@LOAD
This routine loads named routines for application management. The parameter that is passed contains the following:

Name_addr	Fullword Address of the name of module to load (input parameter)
Name_length	Fixed Binary(31) length of module name (input parameter)
User_word	Pointer to a fullword user field (input parameter)
Rsvd_word	Fullword reserved for future use (input parameter). This must be specified as zero (0) for Language Environment/370 Version 1.
Entry_point	Fullword entry point address of the loaded routine (output parameter
Module_size	Fixed Binary(31) size of module that was loaded (output parameter)
Return code	Fullword return code from load (output)
Reason code	Fullword reason code from load (output)

The return/reason codes are listed in Table 34.

Table 34. Return/reason Codes

Return Code	Reason Code	Description
0	0	Successful
0	4	Successful — found as a CMS nucleus extension
0	8	Successful — loaded as a CMS shared segment
0	12	Successful — loaded using SVC8
4	4	Unsuccessful — module loaded above the line when in AMODE(24)
8	4	Unsuccessful — load failed
16	4	Unsuccessful — uncorrectable error occurred

@DELETE
This routine deletes routines for application management. The parameter that is passed contains the following:

Name_addr	Fullword address of the module name to be deleted (input parameter)
Name_length	Fixed Binary(31) length of module name (input parameter)
User_word	Pointer to a fullword user field (input parameter)
Rsvd_word	Fullword reserved for future use. Must be zero. (input parameter)
Return code	Return code from delete service (output)
Reason code	Reason code from delete service (output)

The return/reason codes are listed in Table 35 on page 200.

Table 35. Return/reason Codes

Return Code	Reason Code	Description
0	0	Successful
8	4	Unsuccessful — delete failed
16	4	Unsuccessful — uncorrectable error occurred

@GETSTORE

This routine allocates storage on behalf of the storage manager. This routine can rely upon the caller to provide a save area, which can be the **@Workarea**. The parameter list that is passed contains the following:

Amount Fixed Binary(31) amount of storage requested (input parameter)

Subpool_no Fixed Binary(31) subpool number 0-127 (input parameter)

User word Pointer to a fullword user field (input parameter)

Flags Fullword flag area (input parameter)

Bit zero in the flags is ON if the storage is required below the 16M line. The remaining bits are reserved for future use and must be zero. Bit zero in the flags is OFF if the storage required can be allocated anywhere.

Stg_address Fullword address of the storage obtained or zero (output parameter)

Obtained Fixed Binary(31) number of bytes obtained (output parameter)

Return code Return code from @GETSTORE service (output parameter)

Reason code Reason code from the @GETSTORE service (output parameter)

The return/reason codes are listed in Table 36.

Table 36. Return/reason Codes

Return Code	Reason Code	Description
0	0	Successful
16	0	Unsuccessful — uncorrectable error occurred

@FREESTORE

This routine frees storage on behalf of the storage manager. The parameter list passed contains the following:

Amount Fixed Binary(31) amount of storage to free (input parameter).

Subpool_no Fixed Binary(31) subpool number 0-127 (input parameter).

User word Pointer to a fullword user field (input parameter)

Stg_address Fullword address of the storage to free (input parameter)

Return code Return code from the @FREESTORE service (output)

Reason code Reason code from the @FREESTORE service (output)

The return/reason codes are listed in Table 37.

Table 37. Return/reason Codes

Return Code	Reason Code	Description
0	0	Successful
16	0	Unsuccessful — uncorrectable error occurred

@MSGRTN

This routine allows error messages to be processed by the caller of the application.

Message	A pointer to the first byte of text that will be printed, or zero (input parameter)
Msg_len	A fixed binary(31) containing the length of the message (input parameter)
User word	Pointer to a fullword user field (input parameter)
Line_length	A Fixed Binary(31) size of the output line length. This is used when the **Message** pointer is zero (input parameter).

Return/ Reason code Two fullwords containing the return/reason code

The return/reason codes are listed in Table 38.

Table 38. Return/reason Codes

Return Code	Reason Code	Description
0	0	Successful
16	4	Unsuccessful — uncorrectable error occurred

Chapter 29. Nested Enclaves

An enclave is a logical run-time structure that supports the execution of a collection of routines (see Chapter 2, "Program Model" on page 4 for a detailed description of LE/370 enclaves).

Language Environment/370 Version 1 generally supports the execution of a single enclave within an LE/370 process. Under some circumstances, however, an enclave may create a new enclave and initiate its execution within the same process. This is done under CICS using EXEC CICS LINK and EXEC CICS XCTL (see Chapter 21, "CICS Considerations" on page 120 for more information about these commands), and under MVS, using SVC 6 LINK. There is no CMS nested enclave support under Language Environment/370 Version 1.

LE/370 defines a nested enclave boundary as one that occurs whenever an EXEC CICS LINK, EXEC CICS XCTL, or SCV6 LINK is issued. Whenever a new enclave is created by these commands, the target routine must be a main routine. A link to a subroutine by these commands is not supported under LE/370.

Recursion in a main routine, if supported by its HLL, must be performed without starting a new enclave.

COBOL Considerations: In a non-CICS environment, OS/VS COBOL routines are supported in a single enclave only.

See the *IBM SAA AD/Cycle COBOL/370 Programming Guide* for more information.

Additional Nested Enclave Considerations

Run-time options

When a new enclave is started by EXEC CICS LINK, EXEC CICS XCTL, or SVC 6 LINK, run-time options are inherited from the creating enclave.

Assembler User Exit

An assembler user exit is driven for enclave initialization and enclave termination regardless of whether the enclave is the first enclave created in the process or a nested enclave. The assembler user exit can easily differentiate between first and nested enclave initialization.

Condition Handling

Enclaves created by SVC 6 LINK always propagate conditions of severity 2 or above. The invoked enclave is terminated and the condition is passed to the invoking enclave. The condition is then handled as if the condition occurred in the invoking enclave.

CICS Considerations: Under CICS, enclaves created by EXEC CICS XCTL and EXEC CICS LINK do *not* propagate conditions of severity 2 or above back to the creating enclave. When severity 2 or above conditions occur, the nested enclave is terminated with an ABEND, but no condition token is passed back to the creating enclave. It is instead signaled back to the operating system.

© Copyright IBM Corp. 1991

TRAP(ON/OFF) Effects

In a non-CICS environment, if you specify the run-time option TRAP(ON) in an application that creates a nested enclave, the target enclave inherits TRAP(ON) from the creating enclave. If an ABEND or program interrupt occurs in the created enclave, it is converted to an LE/370 condition. If the condition is unhandled in the created enclave, it is propagated back to the creating enclave. The condition created by the ABEND or program interrupt can then be handled by normal LE/370 condition handling actions in the creating enclave.

If you specify the run-time option TRAP(OFF) in an application that creates a nested enclave, the target enclave inherits TRAP(OFF) from the creating enclave. If an ABEND or program interrupt occurs in the created enclave, TRAP(OFF) prevents the LE/370 condition handler from being notified of the condition. The LE/370 condition handler never gains control in any enclave. The condition is propagated back to the operating system. In addition, all nested enclaves are immediately terminated.

Message File

The message file is not closed when control returns from a nested enclave.

C/370 Considerations: In a non-CICS environment, nested enclaves initiated with an SVC 6 LINK are restricted from using memory files or standard streams.

Using Run-time Options

This section describes options you can specify that control the run-time environment in which your program executes.

Chapter 30. Specifying Run-time Options 206
Order of Precedence 207
Specifying Run-time Options and Program Arguments 207
  CEEXOPT Invocation Syntax 209

Chapter 31. Run-time Options 213
Quick Reference of LE/370 Run-time Options 213
LE/370 Run-time Options 216
  ABPERC 216
  AIXBLD|NOAIXBLD 217
  ALL31 218
  ANYHEAP 219
  ARGPARSE|NOARGPARSE 220
  BELOWHEAP 221
  CBLOPTS 222
  CBLPSHPOP 223
  CBLQDA 223
  CHECK 224
  COUNTRY 224
  DEBUG|NODEBUG 227
  ENV 228
  ERRCOUNT 229
  EXECOPS|NOEXECOPS 229
  FLOW 230
  HEAP 230
  INTERRUPT 232
  LIBSTACK 233
  MSGFILE 234
  MSGQ 235
  NATLANG 235
  PLIST 236
  REDIR|NOREDIR 238
  RPTOPTS 239
  RPTSTG 240
  RTEREUS 243
  SIMVRD 244
  STACK 245
  STORAGE 246
  TERMTHDACT 249
  TEST|NOTEST 250
  TRAP 252
  UPSI 253
  VCTRSAVE 253
  XUFLOW 254

Chapter 32. Language Run-time Option Mapping 255

© Copyright IBM Corp. 1991

Chapter 30. Specifying Run-time Options

LE/370 run-time options can be specified in several ways:

- **Installation Defaults**

  A file, CEEDOPT ORIGINAL, establishes installation defaults using the CEEXOPT macro. The file initially contains IBM-supplied default values for each of the LE/370 run-time options. The syntax for CEEXOPT is presented in the section, "CEEXOPT Invocation Syntax" on page 209. During installation of LE/370, the default values contained in the file can be edited and assembled to create the CEEDOPT CSECT object module. All applications that run in the common run-time environment operate using these default values for the run-time options. The CEEDOPT CSECT resides in the SCEERUN load library.

  It is possible to associate a "nonoverrideable" attribute with each individual run-time option. Each option in CEEDOPT must be specified as either overrideable (OVR) or nonoverrideable (NONOVR). This allows the installation to enforce options that are critical to the overall LE/370 operating environment.

  Note that LE/370 provides the CEECOPT CSECT to establish installation defaults for run-time options under CICS.

  For more information, see *Language Environment/370 Planning for Installation and Customization.*

- **Application defaults**

  Users of LE/370 are also provided with another assembler source file, CEEUOPT ASSEMBLE, setting application defaults for the run-time options using the CEEXOPT macro. Like CEEDOPT, CEEUOPT ASSEMBLE can be edited and assembled to create an object module, CEEUOPT CSECT, that can be linked with an application.

  As noted above, CEEDOPT establishes installation defaults using the CEEXOPT macro. When the program is executed, however, the options specified in CEEUOPT override any corresponding overrideable CEEDOPT options.

  CEEUOPT must be linked with your application in order to establish application defaults.

- **In the MVS JCL**

  Run-time options can be specified with the PARM parameter in the JCL EXEC statement. See "Specifying Run-time Options in the EXEC Statement" on page 42 for details.

- **In TSO commands, on application invocation**

  You can specify run-time options as options on the CALL command.

  See Chapter 13, "Running Your Application under TSO" on page 48 for more information.

- **In CMS commands, on application invocation**

  You can specify run-time options as options on the START and OSRUN commands. They can also be specified when you execute a module produced by the GENMOD command.

  See Chapter 15, "Running Your Application under CMS" on page 56 for more information.

© Copyright IBM Corp. 1991

- **In the assembler user exit**

  See Chapter 27, "Advanced User Exit Topics" on page 173 for information about how to specify a list of run-time options in the assembler user exit.

- **In your source code** (C/370)

  Only C/370 users can specify run-time options in source code using the `#pragma runopts` directive.

  You must specify `#pragma runopts` in the source file that contains your main function. `#pragma runopts` must appear before the first C statement in your source file. Only comments and other pragmas can precede `#pragma runopts`.

  For more information about using C/370 pragmas, see the *IBM SAA AD/Cycle C/370 Programming Guide.*

Syntax

```
                            ┌─,──────┐
                            ▼        │
▶▶──#─pragma─runopts──(───options────┴──)──────────────────────▶◀
```

where *options* is an LE/370 run-time option.

Order of Precedence

It is possible for all the methods listed above to be used for a given application. The order of precedence (from highest to lowest) between option specification methods is:

1. Options defined at installation time that have the nonoverrideable (NONOVR) attribute.
2. Options specified by the assembler user exit.
3. Options specified on invocation.
4. Options provided by the programmer for the application or specified within the source program (that is, link-edited with the application).

   However, you will get a link-edit error if you try to link CEEUOPT to your application at the same time you have specified run-time options using `#pragma runopts`. You may use CEEUOPT **or** `#pragma runopts`, but not both.
5. Option defaults defined at installation time.

Specifying Run-time Options and Program Arguments

In order to distinguish run-time options from program arguments that are passed to LE/370, the options and program arguments are separated by a slash (/). The possible combinations are presented in Table 39 on page 208.

Run-time options usually precede program arguments whenever they are specified in JCL or on application invocation (except for COBOL applications — see below). The possible combinations are described in Table 39 on page 208.

Possible combinations	Format
Both run-time options and program arguments are present	run-time options/program arguments
Only run-time options are present	run-time options/
Only program arguments are present • if a slash is present in the arguments, a preceding slash is mandatory. • if a slash is NOT present in the arguments, a preceding slash is optional.	 /program arguments program arguments **OR** /program arguments

Table 39. Formats for Specifying Run-time options and Program Arguments

You can use the callable service CEE3PRM ("CEE3PRM — Query Parameter String" on page 416) to retrieve program arguments.

Use commas to separate suboptions of run-time options. If you do not specify a suboption, you must still specify the comma to indicate its omission, for example `STACK(,,ANYWHERE,FREE)`. However, trailing commas are not required; `STACK(4K,4K,ANYWHERE)` is valid. If you do not specify *any* suboptions, the following syntax is valid: `STACK()`.

COBOL compatibility considerations: VS COBOL II supports an order of run-time options and program arguments that is the reverse of that expected by LE/370; that is, program arguments precede run-time options in COBOL. For example:

```
program arguments/run-time options
```

To ensure compatibility with COBOL, LE/370 provides the run-time option CBLOPTS. CBLOPTS permits you to choose whether run-time options or program arguments are expected first in the parameter list. You can specify a slash as part of the program arguments with CBLOPTS(ON) or CBLOPTS(OFF). When CBLOPTS(ON) is specified, the last slash in a string delineates the user parameters from the run-time options. Anything before the last slash is interpreted as a user parameter. Conversely, when CBLOPTS(OFF) is specified, the first slash delineates the run-time options from the user parameters. Anything after the first slash is interpreted as a user parameter. CBLOPTS is honored *only* when a COBOL routine is the main routine in the application. If the main is C/370, LE/370 does not honor CBLOPTS. See "CBLOPTS" on page 222 for more information.

C compatibility considerations: If the main routine is C/370, you can specify run-time options at run-time only if you also specify `#pragma runopts(EXECOPS)` in the application. Otherwise, if the NOEXECOPS run-time option is in effect, what you specify on the command line will be passed as program arguments to the main routine.

See "EXECOPSINOEXECOPS" on page 229 for a description of the EXECOPS run-time option.

CEEXOPT Invocation Syntax

Use the CEEXOPT macro to establish installation and programmer default options.

- When invoked during the assembly of CEEDOPT at installation time, CEEXOPT creates the CEEDOPT CSECT, which establishes installation default options. LE/370 run-time options (except those that are C-specific) must be specified in CEEDOPT. Each option in CEEDOPT must be designated as either overrideable (OVR) or nonoverrideable (NONOVR). In addition, a valid value must be specified for each suboption of each run-time option.
- The CEEXOPT macro also creates the CEEUOPT CSECT when CEEUOPT is assembled. CEEUOPT may be linked with an application program to establish user default options. Options in CEEUOPT's invocation of CEEXOPT must *not* be designated as overrideable or nonoverrideable. However, their suboption values take precedence over those of any corresponding overrideable CEEDOPT option values.

To invoke CEEXOPT, adhere to the syntax of the IBM-supplied templates CEEDOPT and CEEUOPT (see Figure 66 and Figure 67 on page 210).

```
CEEDOPT  CSECT
CEEDOPT  AMODE ANY
CEEDOPT  RMODE ANY
         CEEXOPT ABPERC=((NONE),OVR),                                  X
               AIXBLD=((OFF),OVR),                                     X
               ALL31=((OFF),OVR),                                      X
               ANYHEAP=((32K,16K,ANYWHERE,FREE),OVR),                  X
               BELOWHEAP=((32K,16K,FREE),OVR),                         X
               CBLOPTS=((ON),OVR),                                     X
               CBLPSHPOP=((ON),OVR),                                   X
               CBLQDA=((ON),OVR),                                      X
               CHECK=((ON),OVR),                                       X
               COUNTRY=((US),OVR),                                     X
               DEBUG=((ON),OVR),                                       X
               ERRCOUNT=((20),OVR),                                    X
               HEAP=((64K,64K,ANYWHERE,KEEP,16K,16K),OVR),             X
               INTERRUPT=((OFF),OVR),                                  X
               LIBSTACK=((32K,16K,FREE),OVR),                          X
               MSGFILE=((SYSOUT),OVR),                                 X
               MSGQ=((15),OVR),                                        X
               NATLANG=((ENU),OVR),                                    X
               NOTEST=((ALL,*,PROMPT,INSPPREF),OVR),                   X
               RPTOPTS=((OFF),OVR),                                    X
               RPTSTG=((OFF),OVR),                                     X
               RTEREUS=((OFF),OVR),                                    X
               SIMVRD=((OFF),OVR),                                     X
               STACK=((512K,512K,BELOW,KEEP),OVR),                     X
               STORAGE=((NONE,NONE,NONE,8K),OVR),                      X
               TERMTHDACT=((MSG),OVR),                                 X
               TRAP=((ON),OVR),                                        X
               UPSI=((00000000),OVR),                                  X
               VCTRSAVE=((OFF),OVR),                                   X
               XUFLOW=((OFF),OVR)
         END
```

Figure 66. Sample Invocation of CEEXOPT within CEEDOPT ORIGINAL. This is the IBM-supplied version of CEEDOPT, showing the default suboption values for each of the options.

```
CEEUOPT  CSECT
CEEUOPT  AMODE ANY
CEEUOPT  RMODE ANY
         CEEXOPT ABPERC=(NONE),                                        X
               AIXBLD=(OFF),                                           X
               ALL31=(OFF),                                            X
               ANYHEAP=(32K,16K,ANYWHERE,FREE),                        X
               BELOWHEAP=(32K,16K,FREE),                               X
               CBLOPTS=(ON),                                           X
               CBLQDA=(ON),                                            X
               CBLPSHPOP=(ON),                                         X
               CHECK=(ON),                                             X
               COUNTRY=(US),                                           X
               DEBUG=(ON),                                             X
               ERRCOUNT=(20),                                          X
               HEAP=(64K,64K,ANYWHERE,KEEP,16K,16K),                   X
               INTERRUPT=(OFF),                                        X
               LIBSTACK=(32K,16K,FREE),                                X
               MSGFILE=(SYSOUT),                                       X
               MSGQ=(15),                                              X
               NATLANG=(ENU),                                          X
               NOTEST=(ALL,*,PROMPT,INSPPREF),                         X
               RPTOPTS=(OFF),                                          X
               RPTSTG=(OFF),                                           X
               RTEREUS=(OFF),                                          X
               SIMVRD=(OFF),                                           X
               STACK=(512K,512K,BELOW,KEEP),                           X
               STORAGE=(NONE,NONE,NONE,8K),                            X
               TERMTHDACT=(MSG),                                       X
               TRAP=(ON),                                              X
               UPSI=(00000000),                                        X
               VCTRSAVE=(OFF),                                         X
               XUFLOW=(OFF)
         END
```

Figure 67. Sample Invocation of CEEXOPT within CEEUOPT ASSEMBLE. This is the IBM-supplied version of CEEUOPT, showing the default suboption values for each of the options

Notes on CEEXOPT Invocation

1. A continuation character (X in the source) must be present in column 72 on each line of the CEEXOPT invocation except the last line. This applies to both CEEUOPT and CEEDOPT.

2. Options and suboptions must be specified in upper case. Only suboptions that are strings may be specified in mixed or lower case. For example, both MSGFILE=(SYSOUT) and MSGFILE=(sysout) are acceptable. ALL31(off) is not.

3. If one of the string suboptions contains a special character, for example an embedded blank or unmatched right or left parenthesis, the string must be enclosed in apostrophes (''), not in quotation marks (""). (A null string may be specified with either contiguous apostrophes or contiguous quotation marks.)

   To obtain a single apostrophe (') or a single ampersand (&) within a string, two contiguous instances of the character must be specified. The pair is counted as only one character in determining whether the maximum allowable string length has been exceeded, and in setting the effective length of the string.

4. Macro instruction operands may not exceed 255 characters in length. Therefore, it is not possible for each suboption of the TEST or NOTEST options to attain the maximum allowable length normally permitted by LE/370. For example, the *command* suboption of TEST permits 250 characters while the *preference_file* suboption of TEST allows 80. The total number of characters

for these two suboptions therefore exceeds that allowed by the CEEXOPT macro. See "TEST|NOTEST" on page 250 for further information.

If the number of characters to the right of the equal sign is greater than 255 for any keyword parameter in the CEEXOPT invocation in CEEUOPT or CEEDOPT, a return code of 12 is produced for the assembly, and none of the options are parsed properly.

5. Avoid unmatched apostrophes in any string. The error cannot be captured within CEEXOPT itself; instead, the assembler produces a message such as

```
IEV063  *** ERROR ***  NO ENDING APOSTROPHE
```

bearing no particular spatial relationship to the suboption in which the apostrophe was omitted. Furthermore, none of the options are properly parsed if this error is committed.

6. It is possible to completely omit the specification of any option in CEEUOPT. Default values are then supplied for each of the missing suboptions in the Options Control Block that is generated, and these values are ignored at the time LE/370 merges the options.

There are two recommended ways of omitting an option. The HEAP run-time option is used below to demonstrate:

- Specify the option with only a comma following the equal sign:

```
               HEAP=,                                                  X
```

or

- Specify the option with empty parentheses and comma following the equal sign:

```
               HEAP=(),                                                X
```

In either case, the continuation character (X here) must still be present in column 72.

7. In CEEUOPT, it is possible to use commas to indicate the omission of one or more suboptions for options having more than one suboption. For example, if you wish to specify only the second suboption of the STORAGE option, the omission of the 1st, 3rd, and 4th suboptions could be indicated in any of the following ways:

```
               STORAGE=(,NONE),                                        X
               STORAGE=(,NONE,),                                       X
               STORAGE=(,NONE,,),                                      X
```

Because suboptions are positional parameters, do not omit the comma if the corresponding suboption is omitted and another suboption follows.

8. Options need not enclose a single suboption in parentheses. For example, the

COUNTRY option may be specified in CEEUOPT in either of the following ways:

```
          COUNTRY=(US),                                              X
          COUNTRY=US,                                                X
```

Chapter 31. Run-time Options

This section describes:

- **LE/370 run-time options**

  These options may be specified for any LE/370-conforming HLL application.

  Unless otherwise noted, the options can be specified in any of the ways described in Chapter 30, "Specifying Run-time Options" on page 206.

- **COBOL/370-specific run-time options supported under LE/370**

  Unless otherwise noted, these may be specified in any of the ways described in Chapter 30, "Specifying Run-time Options" on page 206.

- **C/370-specific run-time options supported under LE/370**

  These may only be specified using *#pragma runopts*. See Chapter 30, "Specifying Run-time Options" on page 206 for more information.

Note that IBM-supplied default settings for the options are indicated in the option syntax diagrams or in the descriptions of the suboptions, where applicable. The minimum unambiguous abbreviation (where supported) for each LE/370 option is also indicated in its syntax diagram with capital letters (for example, ABPerc indicates that ABP is the minimum abbreviation, and ANyheap indicates that AN is the minimum abbreviation).

Use commas to separate suboptions of run-time options. If you do not specify a suboption, you must still specify the comma to indicate its omission, for example `STACK(,,ANYWHERE,FREE)`. However, trailing commas are not required; `STACK(4K,4K,ANYWHERE)` is valid. If you do not specify *any* suboptions, the following syntax is valid: `STACK()`.

Quick Reference of LE/370 Run-time Options

Table 40 (Page 1 of 4). Run-time Options Quick Reference

Run-Time Options	Function	Page
`ABPerc [( abcode | NONE )]` (default: NONE)	Percolates a specified ABEND.	216
`AIXBLD | AIX | NOAIXbld | NOAIX` (default: NOAIXbld)	Invokes the Access Method Services (AMS) for VSAM indexed and relative data sets to complete the file and index definition procedures for COBOL routines.	217
`AL131 [( OFF | ON )]` (default: OFF)	Indicates whether an application will run entirely in AMODE(31) or will not run entirely in AMODE(31).	218
`ANyheap [( [init_size] , [incr_size] , [ANYwhere | BELOW] , [KEEP | FREE] )]` (defaults: ANYwhere, FREE)	Controls allocation of library heap storage not restricted to below the 16M line.	219

Table 40 (Page 2 of 4). Run-time Options Quick Reference

Run-Time Options	Function	Page
ARGPARSE NOARGPARSE	Specifies whether arguments on the command line are to be parsed in the usual C/370 format.	220
BElowheap (init_size , incr_size , KEEP / FREE)	Controls allocation of library heap storage below the 16M line.	221
CBLOpts (ON / OFF)	Specifies the format of the parameter string on application invocation when the main routine is COBOL.	222
CBLPshpop (ON / OFF)	Controls whether a CICS PUSH HANDLE and CICS POP HANDLE command are issued when a COBOL subprogram is called.	223
CBLQda (ON / OFF)	Controls COBOL QSAM dynamic allocation.	223
CHeck (ON / OFF)	Indicates whether "checking errors" within an application should be detected.	224
COUNTRy (country_id)	Specifies the default formats for date, time, currency symbol, decimal separator, and the thousands separator based on a country.	224
DEBUG NODEBUG	Activates the COBOL batch debugging features specified by the "debugging lines" or the USE FOR DEBUGGING declarative.	227
ENV (CMS / IMS / MVS)	Specifies the operating system that your C/370 application will be running under.	228
ERrcount (number)	Specifies how many severity 2, 3, or 4 errors are allowed before an application is abnormally terminated.	229
EXECOPS NOEXECOPS	Specifies whether run-time options can be specified on the command line.	229
FLOW FLOW(n) FLOW=n FLOWn NOFLOW	Controls the FLOW output produced by OS/VS COBOL programs.	230
HEAP (init_size , incr_size , ANYwhere / BELOW , KEEP / FREE , initsz24 , incrsz24)	Controls allocation of the heaps.	230
INTerrupt (ON / OFF)	Causes attentions recognized by the host operating system to be recognized by LE/370.	232

Table 40 (Page 3 of 4). Run-time Options Quick Reference

Run-Time Options	Function	Page
LIbstack (init_size , incr_size , KEEP / FREE)	Controls the allocation of the thread's library stack storage.	233
MSGFile (ddname)	Specifies the *ddname* of the run-time diagnostics file.	234
MSGQ (number)	Specifies the number of Instance Specific Information (ISI) blocks allocated on a per thread basis during execution.	235
NAtlang (ENU / UEN / JPN)	Specifies the national language to be used for the run-time environment.	235
PLIST(CICS / CMS / HOST / IMS / MVS / OS / TSO)	Specifies the format of the invocation parameters received by your C application when it is invoked.	236
REDIR / NOREDIR	Specifies whether redirections for `stdin,stderr,` and `stdout` are allowed from the command line.	238
RPTOpts (ON / OFF)	Specifies that a report of the run-time options in use by the application will be generated.	239
RPTStg (ON / OFF)	Specifies that a report of the storage used by the application will be generated.	240
RTEREUS / NORTEREUS	Initializes the run-time environment to be reusable when the first COBOL routine is invoked.	243
SIMVRD / NOSIMVRD	Specifies whether your COBOL routines will use a VSAM KSDS to simulate variable length relative organization data sets.	244
STACK (init_size , incr_size , ANYwhere / BELOW , KEEP / FREE)	Controls allocation of the thread's stack storage.	245
STOrage (heap_alloc_value , heap_free_value , dsa_alloc_value , reserve_size)	Controls the value of storage that is allocated and freed.	246
TERmthdact (QUIET / MSG / TRACE / DUMP)	Sets the level of information produced due to an unhandled error of severity 2 or greater.	249

Table 40 (Page 4 of 4). Run-time Options Quick Reference

Run-Time Options	Function	Page
TESt | <u>NOTest</u> ([ALL | ERROR | NONE] , [commands_file | *] , [PROMPT | NOPROMPT | * | ; | command] , [preference_file | *])	Specifies that a debug tool is to be given control according to the sub-options specified.	250
TRap ([ON | <u>OFF</u>])	Specifies how LE/370 routines will handle error conditions and program interrupts.	252
UPSI ([nnnnnnnn])	Sets the eight UPSI switches on or off. Affects only COBOL routines.	253
Vctrsave ([ON | <u>OFF</u>])	Specifies whether any language in an application will use the vector facility when user-provided condition handlers are called.	253
Xuflow ([ON | <u>OFF</u>])	Specifies whether an exponent underflow should cause a program interrupt.	254

LE/370 Run-time Options

ABPERC

The ABPERC option percolates an ABEND whose code you specify.

Syntax

```
▶▶──ABPerc──┬──────────────────┬──────────────────▶◀
            └─(─┬─abcode─┬─)───┘
                └─NONE───┘
```

abcode
specifies the code number of the ABEND you want to percolate.

abcode can be specified as:

S*hhh* a system ABEND code where ***hhh*** is the hex system ABEND code

U*dddd* a user ABEND code where ***dddd*** is a decimal user-issued ABEND code

Any 4-character string may also be used as an *abcode.*

<u>**NONE**</u>
specifies that all ABENDs should be handled according to HLL condition handling semantics.

The IBM-supplied default is ABPERC(NONE).

Usage Notes

1. The ABPERC run-time option is intended for use as a debug tool that allows the application to run with TRAP(ON) in effect. This provides HLL semantics for everything except one ABEND.
2. Only one ABEND code can be identified with this option. However, note that an ABEND U0000 is interpreted the same as S000.
3. You can specify a list of ABEND codes on the assembler user exit that the condition manager percolates. For more information, see Chapter 27, "Advanced User Exit Topics" on page 173.
4. CICS Consideration — ABPERC is ignored under CICS.

Performance Considerations.

None.

AIXBLDINOAIXBLD

The AIXBLDINOAIXBLD run-time option invokes the Access Method Services (AMS) for VSAM indexed and relative data sets (KSDS and RRDS) to complete the file and index definition procedures for COBOL routines. See the *IBM SAA AD/Cycle COBOL/370 Programming Guide* for more details.

Syntax

```
►►──┬─AIXBLD───┬──────────────────────────────────────────►◄
    ├─AIX──────┤
    ├─NOAIXbld─┤
    └─NOAIX────┘
```

AIXBLD
: invokes the Access Method Services for VSAM indexed and relative data sets.

NOAIXBLD
: does not invoke the Access Method Services for VSAM indexed and relative data sets.

The IBM-supplied default is NOAIXBLD.

Usage Notes

1. AIXBLD follows a different abbreviation convention than other run-time options. Only AIX and NOAIX may be used as abbreviations for AIXBLD and NOAIXBLD.
2. When specifying this option in CEEDOPT or CEEUOPT, the only accepted syntax is AIXBLD(ON) or AIXBLD(OFF). AIXBLD and NOAIXBLD are permitted only on the command line.
3. AIXBLD conforms to the ANSI 1985 COBOL standard.
4. MVS Consideration — If the MSGFILE run-time option is specified, the Access Method Services messages are directed to the MSGFILE *ddname* or to the default SYSOUT.
5. CICS Consideration — This option is ignored under CICS.

Performance Considerations

If you use AIXBLD, your program requires more storage for execution. This may slow execution time. Therefore, use AIXBLD only during application development to build alternate indices. Use NOAIXBLD when you have already defined your VSAM data sets.

ALL31

ALL31(ON) allows LE/370 to take advantage of knowing that there are no AMODE(24) routines, only AMODE(31) routines, in the application. ALL31(ON) specifies that an application will run entirely in AMODE(31).

Syntax

```
►►──ALL31──┬────────────────┬──────────────────────────────►◄
           └─(──┬─ON──┬──)──┘
                └─OFF─┘
```

ON

indicates that no user routines of an LE/370 application are AMODE(24).

With ALL31(ON):

- AMODE switching across calls to LE/370 common run-time routines is minimized. For example, no AMODE switching is performed on calls to LE/370 callable services.
- In COBOL, external data is allocated in unrestricted storage.

OFF

indicates that one or more routines of an LE/370 application is AMODE(24).

With ALL31(OFF):

- AMODE switching across calls to LE/370 common run-time routines is performed. For example, AMODE switching is performed on calls to LE/370 callable services.
- In COBOL, external data is allocated in storage below the 16M line.

The IBM-supplied default is ALL31(OFF).

Usage Notes

1. Storage management, in particular that managed by the STACK and HEAP run-time options, is not implicitly altered by this option. However, you must be aware of your application's requirements for STACK and HEAP storage, that can potentially be allocated above the line while executing in AMODE(24).
2. If you are using the default setting ALL31(OFF), you must also use the default setting STACK(,,BELOW). AMODE(24) routines usually require stack storage to be below the 16M line.
3. CICS Consideration — The default under CICS is ALL31(ON).

Performance Consideration

If your application consists of routines that are AMODE(31), your application may run faster with ALL31(ON) than with ALL31(OFF).

ANYHEAP

The ANYHEAP option controls the allocation of library heap storage that is not restricted to a location below the 16M line. See "Heap Storage" on page 76 for more information about LE/370 heaps.

Syntax

```
▸▸──ANyheap──┬──────────────────────────────────────────────────────────────▸
             └─(─┬───────────┬─,─┬───────────┬─,─┬──────────┬────────────▸
                 └─init_size─┘   └─incr_size─┘   ├─ANYwhere─┤
                                                 └─BELOW────┘
▸───────────────────────────────────────────────────────────────────────▸◂
▸──,─┬──────┬─)─┘
     ├─KEEP─┤
     └─FREE─┘
```

init_size
: determines the minimum initial size of the Anywhere Heap storage. This value can be specified as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 4K.

incr_size
: determines the minimum size of any subsequent increment to the Anywhere Heap area, and is specified in *n*, *n*K, or *n*M bytes of storage. This value is rounded up to the nearest multiple of 4K.

ANYWHERE
: specifies that heap storage can be allocated anywhere in storage. On systems that support bi-modal addressing, storage can be allocated either above or below the 16M line. If there is no available storage above the line, then storage is acquired below the line. On systems that do not support bi-modal addressing, for instance, when VM/ESA is initial program loaded (ipled) in 370 mode, this option is ignored and the heap storage is placed below 16 megabytes. ANYWHERE can be abbreviated to ANY.

BELOW
: specifies that the heap storage must be allocated below the 16M line in storage that is accessible to 24-bit addressing.

KEEP
: specifies that storage allocated to ANYHEAP increments is *not* released when the last of the storage is freed.

FREE
: specifies that storage allocated to ANYHEAP increments is released when the last of the storage is freed.

The IBM-supplied default is ANYHEAP(32K,16K,ANYWHERE,FREE).

Usage Notes

1. The ANYHEAP option *cannot* be "turned off". It is *always* in effect. If ANYHEAP or ANYHEAP(0) is specified, then the IBM-supplied default value of 32K of heap storage is allocated when a call is made to obtain heap storage.

2. No Anywhere Heap storage is allocated until the first call to obtain heap storage is made.

3. CICS Considerations — Under CICS, ANYHEAP assumes the defaults ANYHEAP(4K,4K,ANYWHERE, FREE). Both the initial size and the increment size are rounded up to the nearest multiple of 8 bytes. If ANYHEAP or ANYHEAP(0) is specified, then the default value of 4K is assumed. The maximum initial and increment size for ANYHEAP under CICS is 1 gigabyte (1024M).

Performance Considerations

The ANYHEAP option can be used to improve performance when values are specified that minimize the number of times that the operating system allocates storage. The LE/370 RPTSTG run-time option generates a report of the storage used by the application during execution. It can be used to help determine what values should be specified. See "RPTSTG" on page 240 for more information about the RPTSTG run-time option. For more information about fine tuning your application, see "Tuning the Stacks" on page 75 and "Tuning the Heap" on page 78.

ARGPARSE|NOARGPARSE

Note: This option is restricted to applications in which C/370 is the main routine. It can only be specified using *#pragma runopts*. (See Chapter 30, "Specifying Run-time Options" for more information.)

The ARGPARSE|NOARGPARSE option specifies whether arguments on the command line are to be parsed in the usual C/370 format.

Syntax

```
▸▸──┬─ARGPARSE───┬──────────────────────────────▸◂
    └─NOARGPARSE─┘
```

ARGPARSE
: specifies that arguments given on the command line are to be parsed and given to the main routine, that is, the main() function, in the usual C/370 argument format (that is, `argv`, and `argc`).

NOARGPARSE
: specifies that arguments given on the command line are not parsed, but passed to the main routine as one string. Therefore `argc` will have a value of 2 (two) and `argv[1]` will contain a pointer to the string.

The IBM-supplied default is ARGPARSE.

Usage Note

1. CICS Consideration — This option is ignored under CICS.

Performance Considerations

None.

BELOWHEAP

The BELOWHEAP option controls the allocation of library heap storage that must be below the 16M line. The heap controlled by BELOWHEAP is intended for items such as control blocks used for I/O. See "Heap Storage" on page 76 for more information about LE/370 heaps.

Syntax

```
►►──BElowheap──┬──────────────────────────────────────────────────►
               └─(─┬───────────┬─,─┬───────────┬─,─────────────────►
                   └─init_size─┘   └─incr_size─┘

►─────────────────────────────────────────────────────────────────►◄
►──┬──────┬────)─┘
   ├─KEEP─┤
   └─FREE─┘
```

init_size
: determines the minimum initial size of the Below Heap storage. This value can be specified as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 4K.

incr_size
: determines the minimum size of any subsequent increment to the area below the 16M line, and is specified in *n*, *n*K, or *n*M bytes of storage. This value is rounded up to the nearest multiple of 4K.

KEEP
: specifies that storage allocated to BELOWHEAP increments is *not* released when the last of the storage is freed.

FREE
: specifies that storage allocated to BELOWHEAP increments is released when the last of the storage is freed.

The IBM-supplied default is BELOWHEAP(32K,16K,FREE).

Usage Notes

1. The BELOWHEAP option *cannot* be "turned off". It is *always* in effect. If BELOWHEAP is not specified or if BELOWHEAP(0) is specified, then the IBM-supplied default value of 32K of heap storage is allocated when a call is made to obtain heap storage.

2. No Below Heap storage is allocated until the first call to obtain Below Heap storage is made.

3. CICS Considerations — Under CICS, BELOWHEAP assumes the defaults BELOWHEAP(4K,4K,FREE).

Both the initial size and the increment size are rounded to the nearest multiple of 8 bytes. If BELOWHEAP(0) is specified, then the IBM-supplied default value of 4K is assumed for both *init_size* and *incr_size*. The maximum initial and increment size for BELOWHEAP under CICS is 65,504 bytes.

Performance Considerations

You can use the BELOWHEAP option to improve performance by specifying values that minimize the number of times that the operating system allocates storage. The LE/370 RPTSTG run-time option generates a report of storage used by the application during execution. It can be used to help determine what values should be specified. See “RPTSTG” on page 240 for more information about the RPTSTG run-time option. For more information about fine tuning your application, see “Tuning the Stacks” on page 75 and “Tuning the Heap” on page 78.

CBLOPTS

The CBLOPTS option allows you to specify the format of the parameter string on application invocation when the main routine is COBOL. CBLOPTS specifies whether run-time options or program arguments appear first in the parameter string.

Syntax

```
►►──CBLOpts──┬─────────────────┬──────────────────►◄
             └─(──┬───────┬──)─┘
                  ├─ON────┤
                  └─OFF───┘
```

<u>ON</u>
specifies that the parameter string has program arguments first.

OFF
specifies that the parameter string has run-time options first.

The IBM-supplied default is CBLOPTS(ON).

Usage Notes

1. This option can be specified *only* in CEEUOPT or CEEDOPT at initialization. For more information, see Chapter 30, “Specifying Run-time Options” on page 206.

2. When the ON suboption of CBLOPTS is specified in CEEUOPT or CEEDOPT, the run-time arguments and program arguments specified in the JCL or on the command line are honored in the following order:

   program arguments/run-time options

   This order is the reverse of that normally honored by LE/370.

   This option is used only when the MAIN program is COBOL. Otherwise, LE/370 expects run-time options to precede program arguments.

Performance Considerations

None.

CBLPSHPOP

The CBLPSHPOP option allows you to control whether CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a COBOL (VS COBOL II or COBOL/370) subroutine is called.

Syntax

```
▸▸──CBLPshpop─┬──────────────────┬──────────────────────────▸◂
              └─(──┬─────┬──)────┘
                   ├─ON──┤
                   └─OFF─┘
```

ON

Automatically issues the following when a COBOL subroutine is called:

- an EXEC CICS PUSH HANDLE command as part of the routine initialization
- an EXEC CICS POP HANDLE command as part of the routine termination.

OFF

does not issue CICS PUSH HANDLE and CICS POP HANDLE commands on a call to a COBOL subroutine.

The IBM-supplied default is CBLPSHPOP(ON).

Usage Notes

1. Specify CBLPSHPOP(ON) to avoid compatibility problems when calling COBOL/370 or VS COBOL II subroutines that contain CICS CONDITION, AID, or ABEND condition handling commands.
2. Note that the CBLPSHPOP run-time option can also be set on a transaction by transaction basis by using CEEUOPT. For more information, see Chapter 30, "Specifying Run-time Options" on page 206.

Performance Considerations

If your application calls COBOL subroutines under CICS, your application performance will be better with CBLPSHPOP(OFF) than with CBLPSHPOP(ON).

CBLQDA

The CBLQDA option allows you to control COBOL QSAM dynamic allocation on an OPEN statement.

Syntax

```
▸▸──CBLQda─┬──────────────────┬─────────────────────────────▸◂
           └─(──┬─────┬──)────┘
                ├─ON──┤
                └─OFF─┘
```

ON
: specifies that COBOL QSAM dynamic allocation is permitted.

OFF
: specifies that COBOL QSAM dynamic allocation is not permitted.

The IBM-supplied default is CBLQDA(ON).

Usage Notes

1. CBLQDA does not affect dynamic storage allocation for the message file specified in MSGFILE or the dump file.
2. CICS Consideration — This option is ignored under CICS.

Performance Considerations

None.

CHECK

The CHECK option specifies that "checking errors" within an application should be detected. What is defined as a checking error differs among HLLs. COBOL, for example, flags index, subscript, and reference modification ranges as checking errors.

Syntax

```
▶▶──CHeck─┬──────────────────┬─────────────────────────▶◀
          └─(──┬──────┬──)───┘
               ├─ON───┤
               └─OFF──┘
```

ON
: specifies that run-time checking is performed.

OFF
: specifies that run-time checking is not performed.

The IBM-supplied default is CHECK(ON).

Usage Note

1. C Consideration — C/370 does not honor this option.

Performance Considerations

If your COBOL program was compiled with SSRANGE, and you are not testing or debugging an application, you can improve performance by specifying CHECK(OFF).

COUNTRY

The COUNTRY option allows you to set the default formats for date, time, the currency symbol, the decimal separator, and the thousands separator based on a country that you specify.

Syntax

```
▸▸──COUNTRy──┬─────────────────────┬──────────────────▸◂
             └─(─┬────────────┬─)─┘
                 └─country_id─┘
```

country_id
: a 2-character identifier that indicates to LE/370 the country on which the default settings should be based. See Table 41 for a list of countries and their identifiers.

The IBM-supplied default is COUNTRY(US), with US signifying United States.

Usage Notes

1. Appendix G, "IBM-supplied Country Code Defaults" on page 469 contains a list of default settings based on the current country code.

2. The value of COUNTRY can be set either by the run-time option COUNTRY or by the callable service CEE3CTY. For more information about CEE3CTY, refer to "CEE3CTY — Set Default Country" on page 345.

3. The COUNTRY setting affects the format of the date and time in the reports generated by the RPTOPTS and RPTSTG run-time options. For more information on these run-time options, see "RPTOPTS" on page 239 and "RPTSTG" on page 240.

4. accepted with warning message
   ~~If you specify a nonexistent *country_id*, the IBM-supplied *country_id* default setting is used.~~ NOT TRUE Note that CEEUOPT and CEEDOPT permit the specification of a nonexistent country code, but give a return code of 4 and a warning message.

5. C/370 Considerations — C/370 provides locales that map to German, Spanish, French, US English, and others. They establish default formats for items such as currency symbols. To change the locale, you can use the `setlocale()` library function.

   The settings of `setlocale()` and the COUNTRY run-time option do not affect one another. COUNTRY affects only LE/370 services; `setlocale()` affects only C/370 functions.

   To ensure that everything is set properly for your country, use both COUNTRY and `setlocale()` For more information, see the *IBM SAA AD/Cycle C/370 Programming Guide*.

Performance Considerations

None.

Table 41 (Page 1 of 3). Country Identifiers

ID	Country	ID	Country
AD	Andorra	KW	Kuwait
AE	United Arab Emirates	KY	Cayman Islands
AF	Afghanistan	LB	Lebanon
AG	Antigua	LC	Saint Lucia
AL	Albania	LI	Liechtenstein

Table 41 (Page 2 of 3). Country Identifiers

ID	Country	ID	Country
AN	Netherlands Antilles	LK	Sri Lanka
AO	Angola	LR	Liberia
AR	Argentina	LS	Lesotho
AT	Austria	LU	Luxembourg
AU	Australia	LY	Libya
BB	Barbados	MA	Morocco
BD	Bangladesh	MC	Monaco
BE	Belgium	MG	Madagascar
BF	Burkina Faso (Upper Volta)	ML	Mali
BG	Bulgaria	MO	Macau
BH	Bahrain	MR	Mauritania
BI	Burundi	MT	Malta
BJ	Benin	MU	Mauritius
BM	Bermuda	MW	Malawi
BN	Brunei	MX	Mexico
BO	Bolivia	MY	Malaysia
BR	Brazil	MZ	Mozambique
BS	Bahamas	NA	Namibia
BU	Burma	NC	New Caledonia
BW	Botswana	NE	Niger
CA	Canada	NG	Nigeria
CF	Central African Republic	NI	Nicaragua
CG	Congo	NL	Netherlands
CH	Switzerland	NO	Norway
CI	Ivory Coast	NZ	New Zealand
CL	Chile	OM	Oman
CM	Cameroon	PA	Panama
CN	People's Rep of China	PE	Peru
CO	Colombia	PG	Papua New Guinea
CR	Costa Rica	PH	Philippines
CS	Czechoslovakia	PK	Pakistan
CU	Cuba	PL	Poland
CY	Cyprus	PR	Puerto Rico
DE	Federal Rep of Germany	PT	Portugal
DK	Denmark	PY	Paraguay
DO	Dominican Republic	QA	Qatar
DZ	Algeria	RO	Romania
EC	Ecuador	SA	Saudi Arabia
EG	Egypt	SC	Seychelles
ES	Spain	SD	Sudan
ET	Ethiopia	SG	Singapore
FI	Finland	SE	Sweden

Table 41 (Page 3 of 3). Country Identifiers

ID	Country	ID	Country
FR	France	SL	Sierra Leone
GA	Gabon	SN	Senegal
GB	United Kingdom	SO	Somalia
GH	Ghana	SR	Surinam
GM	Gambia	SU	Soviet Union
GN	Guinea	SY	Syria
GR	Greece	SV	El Salvador
GT	Guatemala	SZ	Swaziland
GW	Guinea-Bissau	TD	Chad
GY	Guyana	TG	Togo
HK	Hong Kong	TH	Thailand
HN	Honduras	TN	Tunisia
HT	Haiti	TR	Turkey
HU	Hungary	TT	Trinidad and Tobago
ID	Indonesia	TW	Republic of China
IE	Ireland	TZ	Tanzania
IL	Israel	UG	Uganda
IN	India	US	United States
IQ	Iraq	UY	Uruguay
IR	Iran	VE	Venezuela
IS	Iceland	VU	Vanuatu
IT	Italy	WS	Western Samoa
JM	Jamaica	YE	Yemen
JO	Jordan	YU	Yugoslavia
JP	Japan	ZA	South Africa
KE	Kenya	ZM	Zambia
KR	Korea, Republic of	ZR	Zaire
		ZW	Zimbabwe

DEBUG|NODEBUG

The DEBUG|NODEBUG option activates the COBOL batch debugging features specified by the USE FOR DEBUGGING declarative. See the *IBM SAA AD/Cycle COBOL/370 Programming Guide* for more details.

Syntax

```
▶▶──┬─DEBUG───┬──────────────────────────────────────────────▶◀
    └─NODEBUG─┘
```

DEBUG

activates the COBOL batch debugging features.

You must have the WITH DEBUGGING MODE clause in your application in

order to compile the debugging sections and debugging lines. DEBUG is the default supplied by IBM.

NODEBUG
suppresses the COBOL batch debugging features.

The IBM-supplied default is DEBUG.

Usage Note

1. When specifying this option in CEEDOPT or CEEUOPT, the only accepted syntax is DEBUG(ON) or DEBUG(OFF). DEBUG and NODEBUG are permitted only on the command line.

Performance Consideration

To improve performance, use this option only while debugging.

ENV

Note: This option is restricted to applications in which C/370 is the main routine. It can only be specified using *#pragma runopts*. (See Chapter 30, "Specifying Run-time Options" for more information.)

The ENV option specifies the operating system that your C/370 application is running under.

Syntax

```
▸▸───ENV──(─┬─CMS─┬─)────────────────────────────────▸◂
            ├─IMS─┤
            └─MVS─┘
```

CMS
specifies that the C/370 application runs in a VM environment.

IMS
specifies that the C/370 application runs in an IMS environment. (Note, however, that you do not need to specify the ENV option if your application is invoked under IMS but does not actually use IMS facilities).

MVS
specifies that the C/370 application runs in an MVS environment.

Usage Notes

1. Note that the ENV option differs from other run-time options in that it does not have a standard default. The default depends on what system (CMS or MVS) compilation occurs under.
2. CICS Consideration — This option is ignored under CICS.

Performance Considerations

None.

ERRCOUNT

The ERRCOUNT option specifies how many severity 2, 3, and 4 conditions are allowed to occur before the enclave ABENDs without raising the *Termination_Imminent* condition. If ERRCOUNT(0), an unlimited number of conditions is tolerated.

Syntax

```
▶▶──ERrcount──┬─────────────────┬──────────────────▶◀
              └─(─┬────────┬─)─┘
                  └─number─┘
```

number
: how many severity 2, 3, and 4 conditions will be allowed to occur during the execution of this enclave. If the number of conditions exceeds *number*, then the enclave ABENDs.

The IBM-supplied default is ERRCOUNT(20).

Usage Notes

1. An unlimited number of severity 0 and 1 conditions is permitted.
2. COBOL Considerations — LE/370 does count severity 1 messages with the *facility_id* IGZ. When the limit is reached, additional severity 1 messages are suppressed.

Performance Considerations

None.

EXECOPS|NOEXECOPS

Note: This option is restricted to applications in which C/370 is the main routine. It can only be specified using *#pragma runopts*. (See Chapter 30, "Specifying Run-time Options" for more information.)

The EXECOPS|NOEXECOPS run-time option specifies whether run-time options can be specified on the command line.

Syntax

```
▶▶──┬─EXECOPS───┬──────────────────────────▶◀
    └─NOEXECOPS─┘
```

EXECOPS
: specifies that run-time options can be entered on the command line

NOEXECOPS
: specifies that run-time options cannot be entered on the command line. Any options on the command line are interpreted as arguments to the application.

The IBM-supplied default is EXECOPS.

Usage Notes

1. If NOEXECOPS is specified, the slash (/) which usually separates arguments from run-time options gets passed as an argument to the application.
2. EXECOPS|NOEXECOPS can affect the format of the inbound argument list passed to your application. See Chapter 3, "Parameter List Formats" on page 10 for more information.
3. CICS Consideration — This option is ignored under CICS.

Performance Considerations

None.

FLOW

The FLOW option controls the FLOW output produced by OS/VS COBOL routines.

Syntax

```
▸▸──┬─FLOW────┬──────────────────────────────────────────▸◂
    ├─FLOW(n)─┤
    ├─FLOW=n──┤
    ├─FLOWn───┤
    └─NOFLOW──┘
```

n specifies the number of procedures traced.

n may be any integer from 1 to 99, inclusive.

The IBM-supplied default is NOFLOW, which suppresses the FLOW output produced by OS/VS COBOL routines.

Performance Considerations

None.

HEAP

Heaps are areas of storage where memory is allocated for user-controlled dynamically allocated variables such as C variables allocated as a result of the `malloc()`, `calloc()`, and `realloc()` functions; and COBOL working storage variables.

The HEAP option allows you to control the allocation of the Initial Heap. HEAP also controls allocation of additional heaps created with the CEECRHP callable service. See "CEECRHP — Create New "Additional Heap"" on page 267 for more information. HEAP also specifies how that storage is to be managed. See "Heap Storage" on page 76 for more information about LE/370 heaps.

Syntax

```
▶▶──HEAP──┬──────────────────────────────────────────────────────────────────▶
          └─(─┬───────────┬─,─┬───────────┬─,─┬─ANYwhere─┬─────────────────▶
              └─init_size─┘   └─incr_size─┘   └─BELOW────┘

▶──────────────────────────────────────────────────────────┬──────────────▶◀
▶─,─┬──────┬─,─┬──────────┬─,─┬──────────┬─)──────────────┘
    ├─KEEP─┤   └─initsz24─┘   └─incrsz24─┘
    └─FREE─┘
```

init_size
: determines the minimum initial allocation of the heap storage. This value may be specified as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 4K.

incr_size
: determines the minimum size of any subsequent increment to the heap storage. This value may be specified as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 4K.

ANYwhere
: specifies that heap storage can be allocated anywhere in storage. On systems that support bi-modal addressing, storage can be allocated either above or below the 16M line. If there is no available storage above the line, then storage is acquired below the line. On systems that do not support bi-modal addressing, for instance, when VM/ESA is initial program loaded (ipled) in 370 mode, this option is ignored and the heap storage is placed below 16 megabytes.

  ANYWHERE can be abbreviated to ANY.

BELOW
: specifies that the heap storage must be allocated below the 16M line in storage that is accessible to 24-bit addressing.

KEEP
: specifies that storage allocated to HEAP increments is *not* released when the last of the storage is freed.

FREE
: specifies that storage allocated to HEAP increments is released when the last of the storage is freed.

initsz24
: determines the minimum initial size of the heap storage that is obtained below the 16M line for applications executing with ALL31(OFF) when these applications specify **ANYWHERE** in the HEAP run-time option. *initsz24* may be specified as *n*, *n*K, or *n*M number of bytes. This value is rounded up to the nearest multiple of 4K.

  initsz24 is applicable to all heaps that are not allocated strictly below the 16M line.

incrsz24
: determines the minimum size of any subsequent increment to the heap area that is obtained below the 16M line for applications executing with ALL31(OFF)

when these applications specify **ANYWHERE** in the HEAP run-time option. *incrsz24* may be specified as *n*, *n*K, or *n*M number of bytes. This value is rounded up to the nearest multiple of 4K.

incrsz24 is applicable to all heaps that are not allocated strictly below the 16M line.

The IBM-supplied default is HEAP(64K,64K,ANYWHERE,KEEP,16K,16K).

Usage Notes

1. The HEAP option *cannot* be "turned off". It is *always* in effect. If HEAP is not specified, then the default value of heap storage is allocated when a call is made to obtain heap storage.
2. No heap storage is allocated until the first call to obtain heap storage is made.
3. Applications executing in AMODE(24) that request heap storage obtain the storage below the 16M line regardless of the setting of **ANYWHERE/BELOW**.
4. COBOL Consideration — The HEAP option can be used to provide some of the function provided by COBOL's space management tuning table.
5. CICS Considerations — If HEAP is not specified or if HEAP(0) is specified, LE/370 uses the IBM-supplied default of HEAP(4K,4K,ANY,KEEP,4K,4K). Both the initial HEAP allocation and HEAP increments are rounded to the next higher multiple of 8 bytes (not 4K bytes).

   Under CICS, when HEAP(,,BELOW) is in effect, the maximum size of a heap segment is 65,504 bytes. If too large a value is specified, the application will fail at the first attempt to allocate heap storage. If HEAP(,,ANYWHERE) is in effect, the maximum size of a heap segment is 1 gigabyte (1024M). Note that these CICS restrictions are subject to change from one release of CICS to another.

Performance Considerations

To improve performance, use the storage report generated by the RPTSTG run-time option as an aid in setting the initial and increment size for HEAP. See “RPTSTG” on page 240 for more information about the RPTSTG run-time option.

INTERRUPT

The INTERRUPT option causes attentions recognized by the host operating system to be recognized by LE/370 after the LE/370 environment has been initialized. The way you request an attention interrupt varies from operating system to operating system. When you request the interrupt, control may be given to your application or to a debug tool.

Syntax

```
►►──INTerrupt──┬───────────────┬─────────────────►◄
               └─(──┬─────┬──)─┘
                    ├─ON──┤
                    └─OFF─┘
```

ON
specifies that attention interrupts will be recognized by LE/370. In addition, if the TEST(ERROR) or TEST(ALL) run-time option has been specified, the interrupt causes the debug tool to receive control.

See "TEST|NOTEST" on page 250 for more information about the TEST run-time option.

OFF
specifies that LE/370 will not recognize attentions.

The IBM-supplied default is INTERRUPT(OFF).

Usage Note

1. CICS Consideration — INTERRUPT is ignored under CICS.

Performance Considerations

None.

LIBSTACK

The LIBSTACK option controls the allocation of the thread's library stack storage. This stack is used by LE/370 and HLL library routines that require save areas below the 16M line.

Syntax

```
▶▶──LIbstack──┬──────────────────────────────────────────────▶
              └─(─┬───────────┬─,─┬───────────┬─,────────────▶
                  └─init_size─┘   └─incr_size─┘

▶──────────────────────────────────────────────────────────▶◀
▶─┬──────┬──)─┘
  ├─KEEP─┤
  └─FREE─┘
```

init_size
determines the size of the initial library stack segment. *init_size* may be specified as *n*, *n*K, or *n*M bytes of storage. *init_size* can be preceded by a minus sign. If a negative number is specified, all available storage minus the amount specified is used for the initial stack segment. The storage is contiguous.

In all supported systems except CICS, an *init_size* of 0 or –0 requests half of the largest block of contiguous storage below the 16M line.

At initialization, LE/370 allocates the storage rounded up to the nearest 4K.

incr_size
determines the minimum size of any subsequent increment to the library stack area.

If *incr_size* is not specified, the IBM-supplied default setting of 16K is used. If *incr_size* = 0, then only the amount of storage needed at the time of the request, rounded up to the nearest 4K, is obtained.

KEEP
specifies that storage allocated to LIBSTACK increments is *not* released when the last of the storage is freed.

FREE
specifies that storage allocated to LIBSTACK increments is released when the last of the storage in the library stack is freed. The initial library stack segment is never released until the enclave terminates.

The IBM-supplied default is LIBSTACK(32K,16K,FREE).

Usage Notes

1. CICS Considerations — Under CICS, the maximum initial and increment size for LIBSTACK below 16M is 65,504 bytes.

   The initial and increment sizes for LIBSTACK are rounded to the next higher multiple of 8 bytes.

   The IBM-supplied default setting for LIBSTACK under CICS is LIBSTACK(4K,4K,FREE).

Performance Considerations

To improve performance, use the storage report generated by the RPTSTG run-time option as an aid in setting the initial and increment size for LIBSTACK. See “RPTSTG” on page 240 for more information about the RPTSTG run-time option. For more information about fine tuning your application, see “Tuning the Stacks” on page 75 and “Tuning the Heap” on page 78.

MSGFILE

The MSGFILE option allows you to specify the *ddname* of the file where all run-time diagnostics and reports generated by the RPTOPTS and RPTSTG run-time options are written.

Syntax

```
▸▸──MSGFile──┬────────────────┬──────────────────────▸◂
             └─(─┬────────┬─)─┘
                 └─ddname─┘
```

ddname
The ddname of the run-time diagnostics file.

The IBM-supplied default is MSGFILE(SYSOUT).

Usage Notes

1. HLL compile-time options may affect whether your run-time output goes to MSGFILE *ddname*.
2. LE/370 does not check the validity of the MSGFILE *ddname*. An invalid *ddname* generates an error condition on the first attempt to issue a message.
3. For more information on issuing messages, including COBOL and C run-time output, see Chapter 36, “Message Handling” on page 327.
4. C/370 Considerations — C/370 perror() messages and output directed to `stderr` will go to the MSGFILE. For more information, see Chapter 36, “Message Handling” on page 327.
5. CICS Consideration — The MSGFILE option is ignored under CICS. Run-time output under CICS is directed instead to a transient data queue named CESE.

Performance Considerations

None.

MSGQ

The MSGQ option specifies the number of Instance Specific Information blocks (ISIs) that are allocated on a per thread basis for use by the application. The ISI contains information that is used by the LE/370 condition manager to identify and react to conditions. This permits the occurrence of nested conditions without the loss of message inserts. For more information on nested conditions, see "Nested Conditions" in Chapter 19, "Condition Management" on page 83.

Syntax

```
▸▸──MSGQ──┬──────────────────────┬──────────────────▸◂
          └─(──┬────────┬──)─────┘
               └─number─┘
```

number
: an integer which specifies the number of ISIs to be maintained per thread within an enclave.

The IBM-supplied default is MSGQ(15).

Usage Notes

1. ISIs are returned for reuse after the associated message has been delivered by CEEMSG. See "CEEMSG — Get, Format, and Dispatch a Message" on page 331 for more information.
2. When an ISI is needed for use and one is not available, the least recently used ISI is taken for reuse.

Performance Considerations

None.

NATLANG

The NATLANG option specifies the initial national language that is to be used for the run-time environment, including error messages, month names, and day of the week names. Message translations are provided for Japanese and (uppercase and mixed case) U.S. English. NATLANG also determines how the message facility formats messages.

Syntax

```
▸▸──NAtlang──┬─────────────────┬──────────────────▸◂
             └─(──┬─ENU─┬──)───┘
                  ├─UEN─┤
                  └─JPN─┘
```

ENU
a 3-character id specifying mixed case U.S. English.

Message text is made up of SBCS (single-byte character set) characters and consists of both upper and lowercase letters.

UEN
a 3-character id specifying uppercase U.S. English.

Message text is made up of SBCS characters and consists of uppercase letters.

JPN
a 3-character id specifying Japanese.

Message text can contain a mixture of SBCS and DBCS (double-byte character set) characters.

The IBM-supplied default is NATLANG(ENU).

Usage Notes:

1. Storage and options reports and dump output are written only in mixed case U.S. English.
2. If you specify a *national_language* that is unavailable on your system, the IBM-supplied default *national_language* is used. For LE/370, this is ENU (mixed case U.S. English).

   CEEUOPT and CEEDOPT permit the specification of an unknown national language code, but give a return code of 4 and a warning message. If an invalid language is specified, the IBM-supplied default ENU (mixed case U.S. English) is used. See Chapter 30, "Specifying Run-time Options" on page 206 for more information.
3. The national language can be set using the NATLANG run-time option or the CEE3LNG callable service using the SET option. One current language is maintained at the enclave level and remains in effect until it is changed by one of the above. For, example, if JPN is specified in the NATLANG run-time option, but ENU is specified subsequently using the CEE3LNG callable service, ENU becomes the current national language. For more information about the CEE3LNG callable service, and a list of languages that are valid but are not officially supported by LE/370 in this release, see "CEE3LNG — Set National Language" on page 348.

Performance Considerations

None.

PLIST

Note: This option is restricted to applications in which C/370 is the main routine. It can only be specified using *#pragma runopts*. (See Chapter 30, "Specifying Run-time Options" for more information.)

The PLIST option specifies the format of the invocation parameters received by your C/370 application when it is invoked. Although the CICS, CMS, IMS, MVS, and TSO suboptions of PLIST are supported for compatibility, it is strongly recommended that you use the HOST or OS suboptions of PLIST.

Syntax

```
▸▸——PLIST-(—┬—CICS—┬—)——————————————————————————————▸◂
            ├—CMS——┤
            ├—HOST—┤
            ├—IMS——┤
            ├—MVS——┤
            ├—OS———┤
            └—TSO——┘
```

CICS
the parameter list received by your C/370 application is assumed to be in a CICS format.

CMS
the parameter list received by your C/370 application is assumed to be in a CMS extended parameter list format.

HOST
the parameter list is a character string. The string is located differently under various systems as follows:

- Under CMS, if invoked by OSRUN, use the string presented in an MVS-like format located by the pointer held in R1
- Under CMS, if not invoked by OSRUN, use the CMS Extended parameter list
- Under TSO, if a CPPL is detected, obtain the string from the command buffer
- Under TSO, if a CPPL is not detected, assume a halfword prefixed string in the MVS-format
- Under MVS, use the halfword prefixed string.

IMS
the parameter list received by your C/370 application is assumed to be in an IMS format.

MVS
the parameter list received by your C/370 application is assumed to be in an MVS format.

OS
the parameter list received by your C/370 application is assumed to be in an OS style.

TSO
the parameter list received by your C/370 application is assumed to be in a CPPL (Command Processor Parameter List) format.

The IBM-supplied default is PLIST(HOST).

Usage Notes

1. See Chapter 3, "Parameter List Formats" on page 10 for more information and for a description of the parameter list formats supported under LE/370.

2. When using the pre LE/370-conforming C/370 interface to pre-initialization, it was necessary to specify PLIST(MVS) in order to flag pre-initialized routines. Although PLIST(MVS) is supported for compatibility, it is recommended that you specify PLIST(OS), when possible, to flag these pre LE/370-conforming pre-initialized routines.

3. CICS Consideration — This option is ignored under CICS.

Performance Considerations

None.

REDIR|NOREDIR

Note: This option is restricted to applications in which C/370 is the main routine. It can only be specified using *#pragma runopts*. (See Chapter 30, "Specifying Run-time Options" for more information.)

The REDIR option specifies whether redirections for `stdin`, `stderr`, and `stdout` are allowed from the command line.

Syntax

```
▶▶──┬─REDIR───┬──────────────────────────────────────▶◀
    └─NOREDIR─┘
```

REDIR
specifies that redirections for `stdin`, `stderr`, and `stdout` are allowed from the command line.

REDIR applies only if ARGPARSE is also specified or defaulted.

NOREDIR
specifies that redirections for `stdin`, `stderr`, and `stdout` are not allowed from the command line.

The IBM-supplied default is REDIR.

Usage Notes

1. For ILC applications, REDIR can be used to direct `printf` output from the application to the MSGFILE so it can be interspersed with output from COBOL routines using the C compile-time option DISPLAY. For details, see "ILC Considerations" on page 114.

2. CICS Considerations — This option is ignored under CICS.

Performance Considerations

None.

RPTOPTS

The RPTOPTS option generates, after the application has executed, a report of the run-time options in use during application execution. The report is directed to the *ddname* specified in the MSGFILE run-time option. See the run-time option "MSGFILE" on page 234 for more information.

Syntax

```
►►──RPTOpts──┬──────────────┬──────────────────────────►◄
             └─(──┬─────┬──)─┘
                  ├─ON──┤
                  └─OFF─┘
```

ON
generates a report of the run-time options in use during the execution of the application.

<u>**OFF**</u>
does not generate a report of the run-time options in use during the execution of the application.

The IBM-supplied default is RPTOPTS(OFF).

Usage Notes

1. The options report will not be generated if your application abnormally terminates.
2. Figure 68 on page 240 shows the sample output when the LE/370 RPTOPTS(ON) run-time option is specified. RPTOPTS(ON) lists the declared run-time options in alphabetical order. The option name and where each option obtained its current setting is shown in the report.
3. The report includes language-specific settings.
4. The LAST WHERE SET column in the report shows the last place where the options were referenced even if no suboptions or a subset of the options were changed.
5. "Default setting" in the report indicates that the option cannot be specified in CEEDOPT or CEEUOPT.
6. "Programmer default" includes any options specified with *#pragma runopts*.

Performance Considerations

This option increases the execution time of the application. Therefore, it should be used as an aid for application development, not regular use.

```
Options Report for Enclave NOP 08/23/91 11:06:29 AM

LAST WHERE SET           OPTION
------------------------------------------------------------------------------
Programmer default         ABPERC(NONE)
Invocation command         AIXBLD
Installation default       ALL31(OFF)
Programmer default         ANYHEAP(30720,15360,ANYWHERE,FREE)
Programmer default         BELOWHEAP(36864,18432,FREE)
Installation default       CBLOPTS(ON)
Assembler user exit        CBLPSHPOP(OFF)
Installation default       CBLQDA(ON)
Assembler user exit        CHECK(OFF)
Installation default       COUNTRY(US)
Invocation command       NODEBUG
Programmer default         ERRCOUNT(30)
Default setting          NOFLOW(99)
Installation default       HEAP(65536,65536,ANYWHERE,KEEP,16384,16384)
Installation default       INTERRUPT(OFF)
Default setting            ISAINC(0,0)
Default setting            ISASIZE(0,0,0)
Installation default       LIBSTACK(32768,16384,FREE)
Installation default       MSGFILE(SYSOUT)
Programmer default         MSGQ(10)
Installation default       NATLANG(ENU)
Programmer default         RPTOPTS(ON)
Installation default       RPTSTG(OFF)
Installation default     NORTEREUS
Invocation command       NOSIMVRD
Programmer default         STACK(262144,131072,BELOW,KEEP)
Installation default       STORAGE(NONE,NONE,NONE,8192)
Programmer default         TERMTHDACT(TRACE)
Installation default     NOTEST(ALL,"*","PROMPT","INSPPREF")
Invocation command         TRAP(OFF)
Installation default       UPSI(00000000)
Assembler user exit        VCTRSAVE(OFF)
Programmer default         XUFLOW(OFF)
```

Figure 68. Options Report Produced by LE/370 Run-time Option RPTOPTS(ON)

RPTSTG

The RPTSTG option generates, after the application has executed, a report of the storage that was used by the application. The report is directed to the *ddname* specified in the MSGFILE run-time option. See the run-time option "MSGFILE" on page 234 for more information.

Syntax

```
▸▸───RPTStg──┬────────────────┬──────────────────────────────▸◂
             └──(──┬─────┬──)─┘
                   ├─ON──┤
                   └─OFF─┘
```

ON

generates a report of the storage used during execution of the application.

<u>OFF</u>

does not generate a report of the storage used during execution of the application.

The IBM-supplied default is RPTSTG(OFF).

Usage Notes

1. The storage report will not be generated if your application abnormally terminates.
2. Figure 69 on page 242 shows a sample report created when the RPTSTG(ON) run-time option is specified. The right-hand column in the sample contains explanatory notes that are not included in an actual storage report.
3. The storage report generated by RPTSTG(ON) shows the number of system-level get storage calls that were required during execution of the application. The report can be used to adjust the application in order to reduce the number of times that the LE/370 storage manager must make requests to acquire storage. For example, the storage report can be used to set appropriate values in the HEAP and STACK *init_size* and *incr_size* fields for allocating storage. This should result in some performance improvements for the application.
4. The information produced when this option is specified can be used to adjust the ANYHEAP, BELOWHEAP, HEAP, LIBSTACK, and STACK run-time options.

Performance Considerations

This option increases the execution time of the application. Therefore, it should be used as an aid for application development, not regular use.

```
 Storage Report for Enclave SAMPLE 09/09/91 2:42:53 PM
<Report heading set by CEERPTH appears here, or a blank line>

   STACK statistics:
     Initial size:                                    524288    Amount of storage
                                                                allocated to the first stack
                                                                segment.
     Increment size:                                  524288    Amount of storage allocated
                                                                to each stack increment.
     Total stack storage used (sugg. initial size):     9896    Maximum amount of user stack
                                                                storage actually used.  It
                                                                is generally the optimal
                                                                "initial stack" size.

     No. successful GETMAINs issued:                       1
     No. successful FREEMAINs issued:                      0    Does not include FREEMAINs issued
                                                                during LE/370 termination.

   LIBSTACK statistics:
     Initial size:                                     32768    Amount of storage allocated
                                                                to the first library stack
                                                                segment.
     Increment size:                                   16384    Amount of storage allocated
                                                                to each library stack
                                                                increment.
     Total stack storage used (sugg. initial size):      416    Maximum amount of user stack
                                                                storage actually used.  It
                                                                is generally the optimal
                                                                "initial stack" size.

     No. successful GETMAINs issued:                       1
     No. successful FREEMAINs issued:                      0    Does not include FREEMAINs issued
                                                                during LE/370 termination.

   HEAP statistics:
     Initial size:                                     65536    Amount of storage allocated to
                                                                the first heap segment.
     Increment size:                                   65536    Amount of storage allocated to
                                                                the first heap increment.
     Total heap storage used (sugg. initial size):   1232821    Maximum amount of user heap
                                                                storage actually used.
     No. successful Get Heap requests:                  1449    Calls to CEEGTST that request
                                                                storage from the User Heap.
     No. successful Free Heap requests:                  100    Calls to CEEFRST that free
                                                                storage in the User Heap.

     No. successful GETMAINs issued:                      29
     No. successful FREEMAINs issued:                      1    Does not include FREEMAINs issued
                                                                during LE/370 termination.

   ANYHEAP statistics:
     Initial size:                                     32768    Amount of storage allocated to the
                                                                first Any Heap segment.
     Increment size:                                   16384    Amount of storage allocated to the
                                                                first Any Heap increment.
     Total heap storage used (sugg. initial size):    165000    Maximum amount of LE/370 Any Heap
                                                                storage actually used.
     No. successful Get Heap requests:                    95    Calls to CEEGTST that request
                                                                storage from the LE/370 Below Heap
     No. successful Free Heap requests:                    9    Calls to CEEFRST that free
                                                                storage in the LE/370 Below Heap.

     No. successful GETMAINs issued:                      11
     No. successful FREEMAINs issued:                      1    Does not include FREEMAINs issued
                                                                during LE/370 termination.
```

```
BELOWHEAP statistics:
  Initial size:                                      32768    Amount of storage allocated to the
                                                              first Below Heap segment.
  Increment size:                                    16384    Amount of storage allocated to the
                                                              first Below Heap increment.
  Total heap storage used (sugg. initial size):          0    Maximum amount of LE/370 Below
                                                              Heap storage actually used.
  No. successful Get Heap requests:                      0    Calls to CEEGTST that request
                                                              storage from the LE/370 Below Heap.
  No. successful Free Heap requests:                     0    Calls to CEEFRST that free
                                                              storage in the LE/370 Below Heap.
  No. successful GETMAINs issued:                        0
  No. successful FREEMAINs issued:                       0    Does not include FREEMAINs issued
                                                              during LE/370 termination.

Additional Heap statistics:

  No. successful Create Heap requests:                   0    No. successful calls to CEECRHP
  No. successful Discard Heap requests:                  0    No. successful calls to CEEDSHP

  Total heap storage used:                               0    Maximum amount of storage used
                                                              at any one time in all the
                                                              Additional Heaps combined.
  No. successful Get Heap requests:                      0    Calls to CEEGTST that request
                                                              storage from the Additional Heaps.
  No. successful Free Heap requests:                     0    Calls to CEEFRST that free
                                                              storage in the Additional Heaps.
  No. successful GETMAINs issued:                        0
  No. successful FREEMAINs issued:                       0    Does not include FREEMAINs issued
                                                              during LE/370 termination.

End of Storage Report
```

Figure 69 (Part 2 of 2). Storage Report Produced by LE/370 Run-time Option RPTSTG(ON)

Note: In the preceding table, the numbers displayed following "No. successful GETMAINs issued" and "No. successful FREEMAINs issued" represent the following:

- On VM/ESA, the number of CMSSTOR OBTAIN and CMSSTOR RELEASE requests, respectively.
- On CICS, the number of EXEC CICS GETMAIN and EXEC CICS FREEMAIN requests, respectively.

RTEREUS

The RTEREUS option implicitly initializes the run-time environment to be reusable when the first COBOL routine is invoked.

Syntax

```
▶▶──┬─RTEREUS───┬──────────────────────────────────────▶◀
    └─NORTEREUS─┘
```

RTEREUS
: initializes the run-time environment to be reusable when the first COBOL routine is invoked.

NORTEREUS
: does not initialize the run-time environment to be reusable when the first COBOL routine is invoked.

The IBM-supplied default is NORTEREUS.

Usage Notes

1. When specifying this option in CEEDOPT or CEEUOPT, the only accepted syntax is RTEREUS(ON) or RTEREUS(OFF). RTEREUS and NORTEREUS are permitted only on the command line. See Chapter 30, "Specifying Run-time Options" on page 206 for more information on CEEDOPT and CEEUOPT.
2. IMS Considerations — RTEREUS is not recommended for use under IMS.
3. CICS Consideration — This option is ignored under CICS.

Performance Considerations

You must change STOP RUN statements to GOBACK statements in order to gain the benefits of RTEREUS. STOP RUN terminates the reusable environment. If you specify RTEREUS, on the next invocation of COBOL the reusable environment will be recreated. Doing this repeatedly degrades performance, because a reusable environment takes longer to create than does a normal environment.

LE/370 also offers pre-initialization support in addition to RTEREUS. See Chapter 28, "Pre-initialization" on page 185 for more information.

SIMVRD

The SIMVRD option specifies whether your COBOL routines will use a VSAM KSDS to simulate variable length relative organization data sets. See the *IBM SAA AD/Cycle COBOL/370 Programming Guide* for more details.

Syntax

```
▶▶──┬─SIMVRD───┬─────────────────────────────────────────▶◀
    └─NOSIMVRD─┘
```

SIMVRD
use a VSAM KSDS to simulate variable length relative organization

NOSIMVRD
do not use a VSAM KSDS to simulate variable length relative organization

The IBM-supplied default is NOSIMVRD.

Usage Notes

1. When specifying this option in CEEDOPT or CEEUOPT, the only accepted syntax is SIMVRD(ON) or SIMVRD(OFF). SIMVRD and NOSIMVRD are permitted only on the command line. See Chapter 30, "Specifying Run-time Options" on page 206 for more information on CEEDOPT and CEEUOPT.
2. SIMVRD conforms to the ANSI 1985 COBOL standard.
3. CICS Consideration — This option is ignored under CICS.

Performance Considerations

None.

STACK

The STACK option allows you to control the allocation of the thread's stack storage. Generally residing in the stack are C automatic variables and temporary work areas for COBOL library routines.

Syntax

```
▶▶──STACK─┬──────────────────────────────────────────────────────────▶
          └─(─┬───────────┬─,─┬───────────┬─,─┬───────────┬─────────▶
              └─init_size─┘   └─incr_size─┘   ├─ANYwhere──┤
                                              └─BELOW─────┘

▶─┬──────────────────────┬───────────────────────────────────────▶◀
  └─,─┬──────┬─)─────────┘
      ├─KEEP─┤
      └─FREE─┘
```

init_size
: determines the size of the initial stack segment. The storage is contiguous. The value of *init_size* is specified as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 4K.

  init_size can be preceded by a minus sign. On systems other than CICS, if a negative number is specified, all available storage minus the amount specified is used for the initial stack segment.

  A size of "0" or "-0" requests half of the largest block of contiguous storage in the region below the 16M line. Behavior under CICS is described in the Usage Notes for this run-time option.

incr_size
: determines the minimum size of any subsequent increment to the stack area. This value may be specified as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is whatever value is larger — *incr_size* or the requested size — rounded up to the nearest 4K (except under CICS).

  If *incr_size* is specified as "0", then only the amount of the storage needed at the time of the request, rounded up to the nearest 4K, is obtained.

  The requested size is the amount of storage a routine needs for a stack frame, or DSA. Suppose the requested size is 9000 bytes, *incr_size* is specified as 8K, and the initial stack segment is full. LE/370 will obtain a 12K stack increment from the operating system to satisfy the request. If the requested size is smaller than 8K, LE/370 obtains an 8K stack increment from the operating system.

ANYwhere
: specifies that stack storage can be allocated anywhere in storage. On systems that support bi-modal addressing, storage can be allocated either above or below the 16M line. If there is no available storage above the line, storage is acquired below the line. On systems that do not support bi-modal addressing, for instance, when VM/ESA is initial program loaded (**ipled**) in 370 mode, this option is ignored and the stack storage is placed below 16 megabytes.

<u>BELOW</u>
: specifies that the stack storage must be allocated below the 16M line, in storage that is accessible to 24-bit addressing.

KEEP
: specifies that storage allocated to STACK increments is *not* released when the last of the storage in the stack increment is freed.

FREE
: specifies that storage allocated to STACK increments is released when the last of the storage in the stack is freed. The initial stack segment is never released until the enclave terminates.

The IBM-supplied default is STACK(512K,512K,BELOW,KEEP).

Usage Notes

1. Storage required for the Common Anchor Area (CAA) and other control blocks is allocated separately from, and prior to, the allocation of the Initial Stack segment and the Initial Heap.
2. Applications executing with ALL31(OFF) must specify STACK(,,BELOW) to ensure that stack storage is addressable by the application.

   Note: LE/370 uses the STACK initial size as specified in the installation defaults or programmer's defaults. LE/370 will not use the STACK initial size if the option is specified or modified in the assembler user exit or specified during debugging. Users who want to tune their run-unit execution through the STACK initial size value must relink-edit or recompile their application when a new value is desired.
3. CICS Considerations — The IBM-supplied default setting for STACK under CICS is STACK(4K,4K,ANYWHERE,KEEP).

   The maximum initial and increment size for CICS below 16M is 65504 bytes. The maximum initial and increment size for CICS above 16M is 1 gigabyte (1204M).

   Both the initial size and the increment size are rounded up to the nearest multiple of 8 bytes.

   If STACK is not specified, then the default value of 4K is assumed. Under CICS, STACK(0), STACK (-0), and STACK (-n) are all interpreted as STACK(4K).

Performance Considerations

To improve performance, use the storage report generated by the RPTSTG run-time option as an aid in setting the initial and increment size for STACK. See "RPTSTG" on page 240 for more information about the RPTSTG run-time option. For more information about fine tuning your application, see "Tuning the Stacks" on page 75 and "Tuning the Heap" on page 78.

STORAGE

The STORAGE option controls the initial content of storage when allocated and freed, and the amount of storage that is reserved for the "out-of-storage" condition. If you specify one of the function-values in the STORAGE run-time option, all allocated storage processed by the function is initialized to that value. Otherwise, it is left uninitialized.

The STORAGE option can be used to identify uninitialized application variables, or prevent the accidental use of previously freed storage. STORAGE is also useful in

data security. For example, storage containing sensitive data can be cleared when it is freed.

Syntax

```
▶▶──STOrage──┬──────────────────────────────────────────────────────────────┬──▶
             └─(─┬──────────────────┬─,─┬─────────────────┬─,─┘
                 └─heap_alloc_value─┘   └─heap_free_value─┘

▶─┬──────────────────────────────────────────────────┬──▶◀
  └──┬─────────────────┬─,─┬──────────────┬─)─┘
     └─dsa_alloc_value─┘   └─reserve_size─┘
```

heap_alloc_value
the initialized value of any HEAP storage allocated by the storage manager. *heap_alloc_value* can be specified as:

- a single character enclosed in quotes (See STORAGE Usage Notes for more details). If a single character is specified, every byte of HEAP storage allocated by the storage manager is initialized to that character's EBCDIC equivalent. For example, if you specify 'a' as the *heap_alloc_value*, HEAP storage is initialized to X'818181...81' or 'aaa...a'.
- two hex digits *without* quotes. If you specify two hex digits, every byte of the allocated HEAP storage is initialized to that value. For example, If you specify FE as the *heap_alloc_value*, HEAP storage is initialized to X'FEFEFE...FE'. A *heap_alloc_value* of 00 initializes HEAP storage to X'0000...00'.
- **NONE**. If you specify **NONE**, the freed HEAP storage is not initialized.

heap_free_value
the value of any HEAP storage freed by the storage manager is overwritten. *heap_free_value* can be specified as:

- A single character enclosed in quotes (See STORAGE Usage Notes for more details). A *heap_free_value* of 'f', for example, overwrites freed HEAP storage to X'868686...86'.; 'B' to X'C2'.
- Two hex digits *without* quotes. A *heap_free_value* of FE overwrites freed HEAP storage with X'FEFEFE...FE'.
- **NONE**. If you specify **NONE**, the allocated HEAP storage is not initialized.

dsa_alloc_value
the initialized value of stack frames from the LE/370 stack. A stack frame is dynamically acquired storage that is composed of a standard register save area and the area available for automatic storage.

If specified, all LE/370 stack storage including automatic variable storage is initialized to *dsa_alloc_value.* Stack frames allocated outside the LE/370 stack are never initialized.

dsa_alloc_value can be specified as:

- A single character enclosed in quotes. (See STORAGE Usage Notes for more details). If a single character is specified, any dynamically acquired stack storage allocated by the storage manager is initialized to that charac-

ter's EBCDIC equivalent. For example, if you specify 'A' as the *dsa_alloc_value*, stack storage is initialized to X'C1'. A *dsa_alloc_value* of 'F' initializes stack storage to X'C6', 'd' to X'84'

- Two hex digits *without* quotes. If you specify two hex digits, any dynamically acquired stack storage is initialized to that value. For example, if you specify FE as the *dsa_alloc_value*, stack storage is initialized to X'FE'. A *dsa_alloc_value* of 00 initializes stack storage to X'00', FF to X'FF'
- **NONE**. If you specify **NONE**, the stack storage is not initialized.

reserve_size
: the amount of storage that you want the LE/370 storage manager to reserve in the event of an out-of-storage condition. For more information about the out-of-storage condition, see STORAGE Usage Notes. The value of *reserve_size* can be specified as *n*, *n*K, or *n*M bytes of storage and is rounded to the nearest 4K.

The IBM-supplied default is STORAGE(NONE,NONE,NONE,8K).

Usage Notes

1. *heap_alloc_value*, *heap_free_value*, and *dsa_alloc_value* may all be enclosed in quotes. If you wish to initialize HEAP storage to the EBCDIC equivalent of a single quote, you must double it within the string deliminated by single quotes, or you must surround it with a pair of double quotes. Both of the following are correct ways to specify a single quote:

   STORAGE('''')

   STORAGE("'")

   Similarly, double quotes must be doubled within a string deliminated by double quotes, or surrounded by a pair of single quotes. The following are correct ways to specify a double quote:

   ```
   STORAGE("""")
   STORAGE('"')
   ```

2. If you specify *reserve_size* as 0, no reserve segment is allocated. Without a reserve segment, if your application runs out of storage, it will ABEND with a return code of 4088 and reason code of 1004.

   On systems other than CICS, if you specify a *reserve_size* that is greater than 0, LE/370 does not immediately ABEND when your application runs out of storage. Instead, when the stack overflows, LE/370 attempts to get another stack segment and add it to the stack. If unsuccessful, LE/370 temporarily adds the reserve stack segment to the overflowing stack, and signals the out-of-storage condition. This will allow a user-written condition handler to gain control and release storage. If while this is happening the reserve stack segment overflows, LE/370 then ABENDs with a return code of 4088 and reason code of 1004.

   You can avoid this by increasing the size of the reserve stack segment with the STORAGE(,,,reserve_size) run-time option. The reserve stack segment is not freed until thread termination.

3. CICS Consideration — The IBM-supplied default setting for STORAGE under CICS is STORAGE(NONE,NONE,NONE,0K).

   The out-of-storage condition is not raised under CICS.

Performance Considerations

Use of this service to control initial values can increase execution time. If a *dsa_alloc_value* is specified, performance is likely to be poor. Therefore, the *dsa_alloc_value* option should be used for debugging only, *not* to initialize automatic variables or data structures.

Use STORAGE(NONE,NONE,NONE) when you are not debugging.

TERMTHDACT

The TERMTHDACT option sets the level of information that is produced when a severity 2 or greater condition is percolated beyond the main routine's stack frame.

Syntax

```
►►──TERmthdact──┬──────────────────────┬──────────────────────►◄
                └─(──┬─QUIET─┬──)─┘
                     ├─MSG───┤
                     ├─TRACE─┤
                     └─DUMP──┘
```

QUIET
: specifies that no message is generated when a thread terminates due to an unhandled condition of severity 2 or greater.

MSG
: specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, a message is generated indicating the cause of the thread's termination.

TRACE
: specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, a message indicating the cause of the thread's termination and a trace of the active routines on the activation stack are generated.

DUMP
: specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, a message indicating the cause of the thread's termination, a trace of the active routines on the activation stack, and an LE/370 dump are generated.

The IBM-supplied default is TERMTHDACT(MSG).

Usage Notes

1. The LE/370 service CEE3DMP is called for the TRACE and DUMP suboptions of TERMTHDACT.

   The following CEE3DMP options are passed for TRACE:
   NOENTRY CONDITION TRACEBACK THREAD(ALL) NOBLOCK
   NOSTORAGE NOVARIABLES NOFILES STACKFRAME(ALL)
   PAGESIZE(60) FNAME(CEEDUMP)

   The following options are passed for DUMP:
   THREAD(ALL) NOENTRY TRACEBACK FILES VARIABLES BLOCK
   STORAGE STACKFRAME(ALL) PAGESIZE(60) FNAME(CEEDUMP)
   CONDITION

For more information about the CEE3DMP service and its parameters, see "CEE3DMP — Generate Dump" on page 421.

2. CICS Consideration — All TERMTHDACT output is written to a transient data queue named CESE.

Performance Considerations

None.

TEST|NOTEST

The TEST|NOTEST option specifies the conditions under which the debug tool assumes control when the user application being initialized is invoked using a debug tool such as the AD/Cycle CODE/370 Debug Tool.

Syntax

```
►►──┬─TESt───┬──┬──────────────────────────────────────────────────►
    └─NOTest─┘  └─(──┬───────┬──,──┬─────────────────┬──,──────────►
                     ├─ALL───┤     ├─commands_file───┤
                     ├─ERROR─┤     └────────*────────┘
                     └─NONE──┘

►──────────────────────────────────────────────────┬───────────────►◄
►──────────┬────────────┬──,──┬─────────────────┬──)─┘
           ├─PROMPT─────┤     ├─preference_file─┤
           ├─NOPROMPT───┤     └────────*────────┘
           ├─────*──────┤
           ├─────;──────┤
           └─command────┘
```

ALL
: specifies that any of the following causes the debug tool to get control even without a defined **AT OCCURRENCE** for a particular condition or **AT TERMINATION**:

- The attention function
- Any HLL condition of severity 1 or above
- Application termination.

ERROR
: specifies that only the following causes the debug tool to get control without a defined **AT OCCURRENCE** for a particular condition or **AT TERMINATION**:

- The attention function
- An HLL-defined error condition of severity 2 or above
- Application termination.

NONE
: specifies that no condition causes the debug tool to get control without a defined **AT OCCURRENCE** for a particular condition or **AT TERMINATION**.

commands_file
: a valid *ddname*, dataset name (MVS), or file name (CMS), specifying the primary commands file for this run. If not specified, all requests for commands go to the user's terminal.

The *commands_file* may be enclosed in single or double quotes if necessary to distinguish it from the rest of the TEST|NOTEST suboption list. It can have a

maximum length of 80 characters. If a dataset name is provided that could be interpreted as a *ddname*, the dataset name must be preceded by a slash (/). The slash and dataset name must be enclosed in quotes.

A primary commands file is required when running in a batch environment.

* (in place of *commands_file*)
: specifies that no *commands_file* is supplied. The terminal, if available, is used as the source of the debug tool commands.

PROMPT
: specifies that the debug tool is to be invoked at LE/370 initialization.

NOPROMPT
: specifies that the debug tool is not to be invoked at LE/370 initialization.

* (in place of PROMPT)
: specifies that the debug tool is not to be invoked at LE/370 initialization; equivalent to NOPROMPT.

; (semicolon — in place of PROMPT/NOPROMPT)
: specifies that the debug tool is to be invoked at LE/370 initialization; equivalent to PROMPT.

command
: a character string that specifies a valid debug tool command. The command list may be enclosed in single or double quotes if necessary to distinguish it from the rest of the TEST parameter list. The list cannot contain DBCS characters. The list can have a maximum of 250 characters.

preference_file
: a valid *ddname*, dataset name (MVS), or file name (CMS), specifying the preference file to be used. A preference file is a type of commands file that allows you to specify settings for your debugging environment. It is analogous to creating a profile for a text editor, or initializing an S/370 terminal session.

The *preference_file* may be enclosed in single or double quotes if necessary to distinguish it from the rest of the TEST parameter list and can have a maximum of 80 characters.

If a dataset name is provided which could be interpreted as a *ddname*, the dataset name must be preceded by a slash (/). The slash and dataset name must be enclosed in quotes.

The IBM-supplied default setting for *preference_file* is INSPPREF.

*** (in place of *preference_file*)**
: specifies that no *preference_file* is supplied.

The IBM-supplied default is NOTEST(ALL,*,PROMPT,INSPPREF).

Usage Notes

1. Parameters can be specified on the NOTEST option. If NOTEST is in effect when the application is given control, it is interpreted as TEST(NONE,,*,). If AD/Cycle CODE/370 is initialized using a CALL CEETEST or equivalent, the initial *test level*, the initial *commands_file*, and the initial *preference_file* are taken from the NOTEST run-time option setting.
2. Parameters of the TEST and NOTEST run-time options are merged as one set of parameters.

3. See the *AD/Cycle CODE/370 Reference Guide* for more details and examples of the TEST run-time option.

Performance Consideration

To improve performance, use this option only while debugging.

TRAP

The TRAP option specifies how LE/370 routines handle error conditions and program interrupts.

Syntax

```
▸▸──TRap──┬─────────────────┬──────────────────────────────▸◂
          └─(──┬───────┬──)─┘
               ├─ON────┤
               └─OFF───┘
```

ON
fully enables the LE/370 condition handler.

OFF
prevents member language condition handlers or handlers registered by CEEHDLR from being notified of ABENDS or program checks.

For more information about the CEEHDLR callable service, see "CEEHDLR — Register User Condition Handler" on page 294.

The IBM-supplied default is TRAP(ON).

Usage Notes

1. This option is intended to replace the STAE run-time option currently offered by COBOL and C and the SPIE option offered by C.
2. C Considerations — If multiple instances of STAE/NOSTAE/SPIE/NOSPIE are encountered, all except the last instance will be ignored.
3. The use of the CEESGL callable service remains unaffected by this option. See "CEESGL — Signal a Condition" on page 311 for more information.
4. When TRAP(OFF) is specified in a non-CICS environment, neither ESPIE nor ESTAE is issued. LE/370 messages for conditions raised by either program interruptions or ABENDs initiated by SVC 13 are not printed. The enclave terminates abnormally if such conditions are raised.
5. CICS Considerations — When TRAP(OFF) is specified in a CICS environment, the standard CICS system action occurs. LE/370 messages for conditions raised by either program interruptions or transaction ABENDs are not printed.

Performance Consideration

TRAP(OFF) may alter the condition-handling semantics of your application.

UPSI

Applications that use COBOL routines can use the UPSI run-time option to set the eight UPSI switches on or off. For more information on how COBOL routines access the UPSI switches, see the *IBM SAA AD/Cycle COBOL/370 Programming Guide*.

Syntax

```
►►──UPSI──┬───────────────────────┬──────────────────────►◄
          └─(──┬────────────┬──)─┘
               └─nnnnnnnn───┘
```

nnnnnnnn

n represents one UPSI switch between 0 and 7, the leftmost *n* representing the first switch. Each *n* can either be 0 (off) or 1 (on).

The IBM-supplied default setting is UPSI(00000000).

Usage Note

1. When you specify this option in CEEDOPT or CEEUOPT, you must specify UPSI with a string of 8 binary-valued flags, for example, UPSI(00000000). UPSI, not followed by a string, is permitted only on the command line.

Performance Considerations

None.

VCTRSAVE

The VCTRSAVE option specifies whether any language in the application uses the vector facility when the user-provided condition handlers are called.

Syntax

```
►►──Vctrsave──┬─────────────────┬──────────────────────►◄
              └─(──┬─────┬──)──┘
                   ├─ON──┤
                   └─OFF─┘
```

ON

a language in the application uses the vector facility when user-provided condition handlers are called.

<u>OFF</u>

no language in the application uses the vector facility when user-provided condition handlers are called.

The IBM-supplied default is VCTRSAVE(OFF).

Performance Considerations

When the condition handler plans to use the vector facility (that is, execute any vector instructions), the entire vector environment has to be saved on every condition and restored upon return to the application code. This extra work can be avoided when not needed. Unless you are running an application under vector hardware, specify VCTRSAVE(OFF).

XUFLOW

The XUFLOW option specifies whether an exponent underflow causes a program interrupt. An exponent underflow is produced when a floating point number becomes too small to be represented.

Syntax

```
▸▸──Xuflow──┬───────────────┬──────────────────────────▸◂
            └─(──┬─────┬─)──┘
                 ├─ON──┤
                 └─OFF─┘
```

ON
: an exponent underflow causes a program interrupt.

OFF
: an exponent underflow does not cause a program interrupt; the hardware takes care of the underflow.

The IBM-supplied default is XUFLOW(OFF).

Performance Considerations

XUFLOW is not for general usage. XUFLOW(ON) may change the condition handling semantics of the HLL or HLLs of your application. See "Using XUFLOW and CEE3SPM to Enable and Disable Hardware Conditions" on page 98 for more information.

Chapter 32. Language Run-time Option Mapping

Under LE/370, there is one set of run-time options. LE/370 parses and merges the run-time options. These options are processed at the enclave level and allow you to control many aspects of the LE/370 environment. Note that options are processed in the first enclave only. Subsequent enclaves created within the same LE/370 process inherit run-time options from the first enclave.

Most options are applicable to both LE/370-enabled member languages, although some are specific to a single language. In addition, although LE/370 assists migration by mapping current HLL options to LE/370's options, the run-time options of a particular HLL product may change in the LE/370-enabled version. However, LE/370 has attempted to maintain run-time options consistently across language products while minimizing required changes within each product.

Tables of pre-LE/370, HLL-specific options and their LE/370 equivalents are provided below. The mapping is performed automatically by LE/370 except where noted.

Table 42 (Page 1 of 2). VS COBOL II and LE/370 Run-time Options Comparison

VS COBOL II	LE/370	Notes
AIXBLD	AIXBLD	This option is processed for compatibility, and affects only COBOL routines. "AIX" is a valid abbreviation.
NOAIXBLD	NOAIXBLD	This option is processed for compatibility, and affects only COBOL routines. "NOAIX" is a valid abbreviation.
DEBUG	DEBUG	This option is processed for compatibility, and affects only COBOL routines.
NODEBUG	NODEBUG	This option is processed for compatibility, and affects only COBOL routines.
FLOW	FLOW	Special processing is performed to maintain compatibility with this OS/VS COBOL run-time option. It affects only OS/VS COBOL routines.
LIBKEEP	none	There is no LE/370 equivalent for this option. It is not processed but produces an informative message.
NOLIBKEEP	none	There is no LE/370 equivalent for this option. It is not processed but produces an informative message.
MIXRES	none	There is no LE/370 equivalent for this option. It is not processed but produces an informative message.
NOMIXRES	none	There is no LE/370 equivalent for this option. It is not processed but produces an informative message.
RTEREUS	RTEREUS	This option is processed for compatibility, and affects only COBOL routines.
NORTEREUS	NORTEREUS	This option is processed for compatibility, and affects only COBOL routines.
SIMVRD	SIMVRD	This option is processed for compatibility, and affects only COBOL routines.
NOSIMVRD	NOSIMVRD	This option is processed for compatibility, and affects only COBOL routines.

© Copyright IBM Corp. 1991

Table 42 (Page 2 of 2). *VS COBOL II and LE/370 Run-time Options Comparison*

VS COBOL II	LE/370	Notes
SPOUT	RPTOPTS(ON) RPTSTG(ON)	This is a compatibility mapping to two LE/370 options; it produces an informative message. It affects all languages in an enclave.
NOSPOUT	RPTOPTS(OFF) RPTSTG(OFF)	This is a compatibility mapping to two LE/370 options; it produces an informative message. It affects all languages in an enclave.
SSRANGE	CHECK(ON)	This is a compatibility mapping to an LE/370 option; it produces an informative message. It affects only COBOL routines. "SSR" is a valid abbreviation.
NOSSRANGE	CHECK(OFF)	This is a compatibility mapping to an LE/370 option; it produces an informative message. It affects only COBOL routines. "NOSSR" is a valid abbreviation.
STAE	TRAP(ON)	This is a compatibility mapping to an LE/370 option; it produces an informative message. It affects all languages in an enclave.
NOSTAE	TRAP(OFF)	This is a compatibility mapping to an LE/370 option; it produces an informative message. It affects all languages in an enclave.
UPSI (nnnnnnnn)	UPSI (nnnnnnnn)	This option is processed for compatibility, and affects only COBOL routines.
WSCLEAR	none	There is no LE/370 equivalent for this option. It is not processed but produces an informative message.
NOWSCLEAR	none	There is no LE/370 equivalent for this option. It is not processed but produces an informative message.

Table 43 (Page 1 of 2). *C/370 and LE/370 Run-time Options Comparison*

C/370	LE/370	Notes
ARGPARSE	ARGPARSE	This option is processed for compatibility, and affects C main routines only. It can only be specified using *#pragma runopts*.
NOARGPARSE	NOARGPARSE	This option is processed for compatibility, and affects C main routines only. It can only be specified using *#pragma runopts*.
ENV	ENV	This option is processed for compatibility, and affects C routines only. It can only be specified using *#pragma runopts*.
EXECOPS	EXECOPS	This option is processed for compatibility, and affects C main routines only. It can only be specified using *#pragma runopts*.
NOEXECOPS	NOEXECOPS	This option is processed for compatibility, and affects C main routines only. It can only be specified using *#pragma runopts*.
HEAP(size, incr, loc)	HEAP(size, incr, loc)	This option maps directly, and affects all languages in an enclave.
ISAINC (maj_task_incr, minor_task_incr)	STACK (, maj_task_incr)	This C/370 option maps partially to the STACK LE/370 option, with the remainder of the sub-options processed for compatibility. It affects all languages in an enclave.
ISASIZE (init_size, sub_task_size, max_tasks)	STACK (init_size)	This C/370 option maps partially to the STACK LE/370 option, with the remainder of the sub-options processed for compatibility. It affects all languages in an enclave. In C/370, the ISASIZE option can be abbreviated to ISA. This abbreviation is supported by LE/370.

Table 43 (Page 2 of 2). C/370 and LE/370 Run-time Options Comparison

C/370	LE/370	Notes
LANGUAGE (national_language)	NATLANG (national_language)	For compatibility with the C/370 option LANGUAGE, LE/370 supports the following national languages: Mixed-case American English (ENGLISH) - maps to the NATLANG(ENU) option. Uppercase American English (UENGLISH) - maps to the NATLANG(UEN) option with separate flag to indicate 'fold to upper'. Japanese (NIHONGO, JAPANESE) - maps to the NATLANG(JPN) option. This option affects all languages in an enclave. The LANGUAGE option may be truncated to LANG. EN is a valid abbreviation for ENGLISH, UE is a valid abbreviation for UENGLISH, NI is a valid abbreviation for NIHONGO, and JA is a valid abbreviation for JAPANESE. All of these abbreviations are supported by LE/370.
PLIST	PLIST	This option is processed for compatibility, and affects C main routines only. It can only be specified using *#pragma runopts.*
REDIR	REDIR	This option is processed for compatibility, and affects C main routines only. It can only be specified using *#pragma runopts.*
NOREDIR	NOREDIR	This option is processed for compatibility, and affects C main routines only. It can only be specified using *#pragma runopts.*
REPORT	RPTSTG(ON)	This is a compatibility mapping to an LE/370 option, and it affects all languages in an enclave. The REPORT option can be abbreviated to R.
NOREPORT	RPTSTG(OFF)	This is a compatibility mapping to an LE/370 option, and it affects all languages in an enclave. The NOREPORT option can be abbreviated to NR.
SPIE	TRAP(ON)	This is a compatibility mapping to an LE/370 option, and it affects all languages in an enclave.
NOSPIE	TRAP(OFF)	This is a compatibility mapping to an LE/370 option, and it affects all languages in an enclave.
STACK	STACK	This is a direct mapping, and it affects all languages in an enclave.
STAE	TRAP(ON)	This is a compatibility mapping to an LE/370 option, and it affects all languages in an enclave.
NOSTAE	TRAP(OFF)	This is a compatibility mapping to an LE/370 option, and it affects all languages in an enclave.
TEST(control, cmds_file, cmd_str)	TEST(control, cmds_file, cmd_str)	This is a direct mapping. The LE/370 TEST option also has a fourth parameter, preference_file. Its current setting remains unchanged. This option affects all languages in an enclave.
NOTEST(control, cmds_file, cmd_str)	NOTEST(control, cmds_file, cmd_str)	This is a direct mapping. The LE/370 NOTEST option also has a fourth parameter, preference_file. Its current setting remains unchanged. This option affects all languages in an enclave.

Using Callable Services

This section describes the use of the LE/370 callable services. Syntax, parameter descriptions, usage notes, and examples are provided for each callable service. The services are grouped by function.

Chapter 33. Invoking Callable Services 261
Data Type Definitions 262
Quick Reference of LE/370 Callable Services 263

Chapter 34. Dynamic Storage Callable Services 267
CEECRHP — Create New "Additional Heap" 267
CEECZST — Reallocate (Change siZe of) STorage 270
CEEDSHP — Discard Heap 274
CEEFRST — Free Heap Storage 276
CEEGTST — Get Heap Storage 278
CEE3RPH — Set Report Heading 281
Examples Using Several Dynamic Storage Services 283

Chapter 35. Condition Handling 287
CEEDCOD — Decompose a Condition Token 287
CEEGPID —Retrieve the LE/370 Version and Platform ID 290
CEEGQDT — Retrieve Q_Data_Token 291
CEEHDLR — Register User Condition Handler 294
CEEHDLU — Unregister User Condition Handler 296
CEEITOK — Return Initial Condition Token 298
CEEMRCR — Move Resume Cursor Relative to Handle Cursor 300
CEENCOD — Construct a Condition Token 306
CEESGL — Signal a Condition 311
CEE3ABD —Terminate Enclave with an ABEND 313
CEE3CNC —Allow Nested Conditions 315
CEE3GRN —Get Name of Routine that Incurred Condition 316
CEE3SPM — Query and Modify LE/370 Hardware Condition Enablement 318
Example Using Several Condition Handling Services 323

Chapter 36. Message Handling 327
CEEMGET — Get a Message 327
CEEMOUT — Dispatch a Message 330
CEEMSG — Get, Format, and Dispatch a Message 331
Examples Using Several Message Handling Services 333

Chapter 37. National Language Support 338
CEEFMDA — Obtain Default Date Format 338
CEEFMDS — Obtain Default Decimal Separator 339
CEEFMDT — Obtain Default Date and Time Format 341
CEEFMTM — Obtain Default Time Format 343
CEE3CTY — Set Default Country 345
CEE3LNG — Set National Language 348
CEE3MCS — Obtain Default Currency Symbol 352
CEE3MTS —Obtain Default Thousands Separator 354
Examples Using Several NLS Services 356

© Copyright IBM Corp. 1991

Chapter 38. Date and Time Services 359
Introduction 359
CEEDAYS — Convert Date to Lilian Format 361
CEEDATE — Convert Lilian Date to Character Format 368
CEEDATM — Convert Seconds to Character Timestamp 371
CEEDYWK — Calculate Day of Week from Lilian Date 374
CEEGMT — Get Current Greenwich Mean Time 376
CEEGMTO — Get Offset From Greenwich Mean Time to Local Time . . . 378
CEEISEC — Convert Integers to Seconds 380
CEELOCT — Get Current Local Time 383
CEEQCEN — Query the Century Window 385
CEESCEN — Set the Century Window 387
CEESECI — Convert Seconds to Integers 389
CEESECS — Converts Timestamp to Number of Seconds 391
CEEUTC — Get Coordinated Universal Time 395
Examples Using Several Date and Time Services 395

Chapter 39. Math Routines 406
Calling the Math Routines 406
Math Routine Descriptions 407
Feedback Code Descriptions 411
COBOL Considerations 413
Examples 413
COBOL/370 Examples 413
C/370 Example 415

Chapter 40. General LE/370 Callable Services 416
CEE3PRM — Query Parameter String 416
CEE3USR — Set or Query User Area Fields 417
CEETEST — Invoke Debug Tool 419
CEE3DMP — Generate Dump 421
CEERAN0 — Calculate Uniform Random Numbers 426

Chapter 41. Guidelines for Writing an LE/370 Callable Service 429

CLAIMS 1-15 ARE PENDING

1. The drawings are objected to on the grounds that FIG 2 does not conform to the claims as amended. In particular, is shows box **36** labeled: reformat data in database. The summary of the invention, the statements at the bottom of page 3 of the Response, and the Reasons for Allowance below specifically preclude this step. This box should be removed.

2. The following is an examiner's statement of reasons for allowance:

The Prior Art of Record, taking into account the Affidavit of the inventor, received 3/24/98, swearing behind the reference of the previous action, does not anticipate nor suggest the set of limitations of the claims, comprising the threshold year digits as used to determine a pair of century digits to be used for computation, but without enlarging the number of date digits of the database.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

Scott Bradshaw
Patent tech.
-Ken Gorell

CLAIMS 1-15 ARE PENDING

1. The disclosure is objected to because of the following informalities:

The statement of the problem, at the bottom of page 1, which implies that requiring additional data fields for storage to solve the Y2K problem, is contradicted by the use of century digits C_1C_2, as spelled out at page 2, second paragraph of the SUMMARY. Similarly, so is the second sentence of the SUMMARY.

Appropriate correction is required.

2. Claims 1-15 are rejected under 35 U.S.C. 112, first paragraph, as based on a disclosure which is not enabling. The "conversion of existing symbolic date representations ... without the addition of new data fields", as indicated at page 2 lines 7-10, is critical or essential to the practice of the invention, but not included in the claim(s) is not enabled by the disclosure. *In re Mayhew*, 527 F.2d 1229, 188 USPQ 356 (CCPA 1976).

The problem set forth in the last four lines of page 1 and promised in the first paragraph of page 2, as well as in the lines quoted above, indicate that the invention solves the Y2K problem without introducing additional digits. The claims, the abstract, and the description of the invention in the SUMMARY clearly involve century digits C_1C_2, which increase the number of date digits from 6 to 8, thus using 4 digits to indicate the year. One of ordinary skill in the art would not know how to resolve this discrepancy.

Goldberg, Gerald

From: Goldberg, Gerald
Sent: Tuesday, January 11, 2000 4:36 PM
To: Mills, John (2700)
Cc: Black, Thomas
Subject: FW: Copy of Transcript

Yup. I will place a copy into Tom Black's inbox for you.

-----Original Message-----
From: Cleveland, Carol
Sent: Tuesday, January 11, 2000 4:28 PM
To: Rolla, Joseph; Goldberg, Gerald
Cc: Mills, John (2700)
Subject: Copy of Transcript

NPR held a program yesterday regarding Y2K patent and John Mills would like to know if he could get a copy of the transcript.

John Mills 308-9822

Thank You

Carol D. Cleveland
Voice Line 703-305-9700
Fax Number 703-308-5355

National Public Radio (NPR)
SHOW: MORNING EDITION (10:00 AM ET)
January 10, 2000, Monday

HEADLINE: SOFTWARE ENGINEER IN SOUTHERN CALIFORNIA WHO PATENTED A SIMPLE Y2K FIX WANTS COMPANIES TO PAY HIM MONEY

ANCHORS: BOB EDWARDS

REPORTERS: ELAINE KORRY

BODY:
BOB EDWARDS, host:

Despite fears of widespread chaos, the New Year passed with only minor disruptions to the nation's computer systems. A software engineer in Southern California claims that a simple Y2K fix he patented averted a crisis. He wants millions of dollars in royalties from every company that used his solution. The high-tech world is in an uproar and the US **Patent and Trademark Office** is reconsidering whether the **patent** is valid. NPR's Elaine Korry reports from San Francisco.

ELAINE KORRY reporting:

All his life, Bruce Dickens dreamed of patenting his own invention. His breakthrough came on a February morning in 1995. The middle-age programmer had been assigned to devise a Y2K fix for a complex inventory control program at McDonnell Douglas. Stuck in commuter traffic that day, an idea suddenly struck Dickens, a quick and easy way to trick a computer into thinking that the millennium date change was decades away.

Mr. BRUCE DICKENS: As soon as I got to work, then, I sat down my computer, started banging away this new method to do it, and sure enough, it just blossomed out in front of me. And I was just exhilarated.

KORRY: Dickens' solution is far less complicated than the alternative many companies faced rewriting millions of lines of software code. Industry analysts say most of the nation's businesses wound up using a fix much like his to correct their millennium bugs. According to Dickens' attorney, William Cray, he deserves to be compensated.

Mr. WILLIAM CRAY (Attorney): Well, initially, the thought was to go after the people who've saved the most amount of money.

KORRY: Cray says that's all of big business. He has contacted 700 major corporations he says saved hundreds of millions of dollars each by using Dickens' solution. He's

demanding they fork over a percentage of their savings as royalties. Few of the firms Cray contacted even replied, and now he's increased his royalty demand 100-fold. Many Fortune 500 executives are privately outraged, but reluctant to speak publicly about the case. Like them, James Boyle, who teaches intellectual property law at American University, questions whether the patent was properly issued in the first place.

Professor JAMES BOYLE (American University): The **Patent and Trademark Office** is notoriously bad at checking software prior art, which is the specialized pen language for saying that they don't go back to find out whether or not people have already used these techniques, in which case, of course, they're not novel, they're not inventions and they don't deserve to be patented.

KORRY: Simply put, Dickens' method tells computers to automatically assign two-digit dates to either the 20th or the 21st century. According to certain preset windows, a concept unrelated to Microsoft Windows. For instance, the year '00 through '29 would be lumped into one window of year 2000 dates, while the years '30 through '99 would fall in the 1900 window.

Harris Miller, the president of the Information Technology Association of America, says Dickens' discovery is nothing new.

Mr. HARRIS MILLER (Information Technology Association of America): We have on our Web site, I believe, over 30 examples that have been given to us by various academics and computer programmers, other experts in the field, of long-standing documentation of people using windowing techniques.

KORRY: But Dickens' attorney, William Cray, says that argument misses the point.

Mr. CRAY: Windowing, as a concept itself, has been around for a long time. Bruce doesn't claim to have invented windowing per se.

KORRY: Cray says Dickens' added refinements and other steps layered on top of the windowing concept that make his contribution truly unique. In light of the outcry, the **Patent and Trademark Office** has decided to take a second look at the prior art. The office re-examines about 400 patents a year, but in this case, commissioner **Todd Dickinson** has taken the unusual step of ordering a re-examination himself. He won't comment on the merits of the Dickens' patent, but he has this response to critics such as law Professor Boyle.

Mr. **TODD DICKINSON (Patent and Trademark Office** Commissioner): Mr. Boyle has art that he thinks we haven't considered, he should send it on in. I will tell you this, I've made this offer many times and I've had very few people take me up on it.

KORRY: If his patent is upheld, Dickens stands to become a billionaire, at least in theory. There are bound to be legal challenges. Dickens has been called an opportunist, but his attorney says he's only doing what patent holders normally do, seek payment for

the use of their inventions. Dickens claims the money is only an afterthought. What he really wants is to see his discovery vindicated.

Mr. DICKENS: It's a model patent, it's an excellent example of a patent, excellent invention, and I'm going to assert my rights.

KORRY: The re-examination process could take as long as two years. In the meantime, Dickens' patent is valid and he will continue to pursue royalties. Elaine Korry, NPR News, San Francisco.

LANGUAGE: English

LOAD-DATE: January 10, 2000

Black, Thomas

From: Ng, Jin
Sent: Friday, February 18, 2000 10:38 AM
To: An, Meng-Ai; Asta, Frank; Au, Amelia; Beausoleil, Robert; Black, Thomas; Bost, Dwayne; Boudreau, Leo; Breene, John; Burgess, Glenton; Cabeca, John; Chan, Eddie; Chan, Jason; Chin, Stephen; Chin, Wellington; Coles, Edward; Decady, Albert; Eisenzopf, Reinhard; Faile, Andrew; Fetting, Anton; Garber, Wendy; Grant, William; Hafiz, Tariq; Hayes, Gail; Hjerpe, Richard; Hofsass, Jeffrey; Horabik, Michael; Hudspeth, David; Hunter, Daniel; Isen, Forester; Kelley, Chris; Kim, Matt; Kizou, Hassan; Kuntz, Curtis; Lee, Thomas; Levy, Stuart; Lintz, Paul; Loomis, Paul; MacDonald, Allen; Matar, Ahmad; Moore, David; Nguyen, Chau; Oberley, Alvin; Olms, Douglas; Palys, Joseph; Peng, John; Pham, Chi; Powell, Mark; Razavi, Michael; Saras, Steven; Shalwala, Bipin; Sheikh, Ayaz; Swann, Tod; Teska, Kevin; Trammell, James; Tsang, Fan; Voeltz, Todd; Vu, Kim; Yoo, Do; Zele, Krista; Zimmerman, Mark
Cc: Dwyer, James; Garrett, Robert; Goldberg, Gerald; Rolla, Joseph
Subject: RE: Management/Supervisory Training

I was informed that there are very limited seats available and that we can do one training form for all attendees. All I need are the names, SSNs and the registration forms to me by 2/24. Thanks.

Jin F. Ng

-----Original Message-----
From: Ng, Jin
Sent: Thursday, February 17, 2000 3:44 PM
To: An, Meng-AI; Asta, Frank; Au, Amelia; Beausoliel, Robert; Black, Thomas; Bost, Dwayne; Boudreau, Leo; Breene, John; Burgess, Glenton; Cabeca, John; Chan, Eddie; Chan, Jason; Chin, Stephen; Chin, Wellington; Coles, Edward; Decady, Albert; Eisenzopf, Reinhard; Faile, Andrew; Fetting, Anton; Garber, Wendy; Grant, William; Hafiz, Tariq; Hayes, Gail; Hjerpe, Richard; Hofsass, Jeffrey; Horabik, Michael; Hudspeth, David; Hunter, Daniel; Isen, Forester; Kelley, Chris; Kim, Matt; Kizou, Hassan; Kuntz, Curtis; Lee, Thomas; Levy, Stuart; Lintz, Paul; Loomis, Paul; MacDonald, Allen; Matar, Ahmad; Moore, David; Nguyen, Chau; Oberley, Alvin; Olms, Douglas; Palys, Joseph; Peng, John; Pham, Chi; Powell, Mark; Razavi, Michael; Saras, Steven; Shalwala, Bippin; Sheikh, Ayaz; Swann, Tod; Teska, Kevin; Trammell, James; Tsang, Fan; Voeltz, Todd; Vu, Kim; Yoo, Do; Zele, Krista; Zimmerman, Mark
Cc: Dwyer, James; Garrett, Robert; Goldberg, Gerald; Rolla, Joesph
Subject: FW: Management/Supervisory Training

FYI. Registration forms are available in my Office.

Jin F. Ng

-----Original Message-----
From: Ng, Jin
Sent: Thursday, February 17, 2000 1:28 PM
To: Patent Directors; Kunin, Stephen

Cc: Patent Corps Secretaries; Godici, Nicholas; Kisliuk, Bruce; Goldberg, Howard
Subject: Management/Supervisory Training

The following training is available from James Madison University's National Center for Professional Development. A registration form has been put in your box on the 9th floor. Copies will also be available at WIL next week. If you have supervisors who want to attend, please have registration forms completed and sent to me by 3/1/00 so I can get the group rate. Funding is based on the available Management Training allocation in each TC/area which was distributed last month. Thanks.

Lessons in Leadership

Date & Time:

Thursday, April 13, 2000, 8:30am - 3:00pm (Seating is open. Doors will open at 7:30am) Lunch will be served from 11:45sm - 1:00pm.

Location:

Omni Shoreham Hotel, 2500 Calvert Street, NW, Washington DC

Speaker:

Dr. Stephen Covey

Program:

"The Four Roles of a Leader: How to Make Every Team Player a Leader"

How to build and sustain an atmosphere of trust and openness
How to use creative cooperation to reach new levels of performance
How to develop leadership at every level of your organization
How to reduce cynicism and increase morale
How to stay flexible and focused to recognize larger opportunities
How to take advantage of strengths and compensate for weaknesses
How to stay more "promotable"
How to keep and inspire your most talented workers

Cost:

30 or more	$339 each
20-29	$349 each
10-19	$$359 each
3-9	$369 each
1-2	$379 each

copy 2

CLAIMS 1-15 ARE PENDING

1. The drawings are objected to on the grounds that FIG 2 does not conform to the claims as amended. In particular, is shows box **36** labeled: reformat data in database. The summary of the invention, the statements at the bottom of page 3 of the Response, and the Reasons for Allowance below specifically preclude this step. This box should be removed.

2. The following is an examiner's statement of reasons for allowance:

The Prior Art of Record, taking into account the Affidavit of the inventor, received 3/24/98, swearing behind the reference of the previous action, does not anticipate nor suggest the set of limitations of the claims, comprising the threshold year digits as used to determine a pair of century digits to be used for computation, but without enlarging the number of date digits of the database.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

3. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Wayne Amsbury whose telephone number is (703) 305-3828. The examiner can normally be reached on Monday-Thursday from 6:30 AM to 5:00 PM Eastern time.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas G. Black, can be reached on (703) 305-9707. The fax phone number for this Art Unit is (703) 305-9731.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-9600.

April 2, 1998

copy 1

CLAIMS 1-15 ARE PENDING

1. The drawings are objected to on the grounds that FIG 2 does not conform to the claims as amended. In particular, is shows box **36** labeled: reformat data in database. The summary of the invention, the statements at the bottom of page 3 of the Response, and the Reasons for Allowance below specifically preclude this step. This box should be removed.

2. The following is an examiner's statement of reasons for allowance:

The Prior Art of Record, taking into account the Affidavit of the inventor, received 3/24/98, swearing behind the reference of the previous action, does not anticipate nor suggest the set of limitations of the claims, comprising the threshold year digits as used to determine a pair of century digits to be used for computation, but without enlarging the number of date digits of the database.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

3. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Wayne Amsbury whose telephone number is (703) 305-3828. The examiner can normally be reached on Monday-Thursday from 6:30 AM to 5:00 PM Eastern time.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas G. Black, can be reached on (703) 305-9707. The fax phone number for this Art Unit is (703) 305-9731.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-9600.

April 2, 1998

Interview Summary

Application No.	Applicant(s)
08/725,574	Dickens
Examiner	**Group Art Unit**
Wayne Amsbury	2771

All participants (applicant, applicant's representative, PTO personnel):

(1) *Wayne Amsbury* (3)

(2) *Guy Gosnell* (4)

Date of Interview *Apr 2, 1998*

Type: ☒ Telephonic ☐ Personal (copy is given to ☐ applicant ☐ applicant's representative).

Exhibit shown or demonstration conducted: ☐ Yes ☒ No. If yes, brief description:

Agreement ☒ was reached. ☐ was not reached.

Claim(s) discussed: *1-15*

Identification of prior art discussed:
Art of record in the case.

Description of the general nature of what was agreed to if an agreement was reached, or any other comments:
It was agreed that the summary of the invention, and the arguments of the response, were not entirely in conformity with the claims, which would be potentially allowable if the use of additional century digits did not include their storage in the database. It was further agreed that Applicant would fax a supplementary amendment to address this problem.

(A fuller description, if necessary, and a copy of the amendments, if available, which the examiner agreed would render the claims allowable must be attached. Also, where no copy of the amendents which would render the claims allowable is available, a summary thereof must be attached.)

1. ☒ It is not necessary for applicant to provide a separate record of the substance of the interview.

Unless the paragraph above has been checked to indicate to the contrary, A FORMAL WRITTEN RESPONSE TO THE LAST OFFICE ACTION IS NOT WAIVED AND MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW. (See MPEP Section 713.04). If a response to the last Office action has already been filed, APPLICANT IS GIVEN ONE MONTH FROM THIS INTERVIEW DATE TO FILE A STATEMENT OF THE SUBSTANCE OF THE INTERVIEW.

2. ☐ Since the Examiner's interview summary above (including any attachments) reflects a complete response to each of the objections, rejections and requirements that may be present in the last Office action, and since the claims are now allowable, this completed form is considered to fulfill the response requirements of the last Office action. Applicant is not relieved from providing a separate record of the interview unless box 1 above is also checked.

Examiner Note: You must sign and stamp this form unless it is an attachment to a signed Office action.

Copy 1

CLAIMS 1-15 ARE PENDING

1. The disclosure is objected to because of the following informalities:

The statement of the problem, at the bottom of page 1, which implies that requiring additional data fields for storage to solve the Y2K problem, is contradicted by the use of century digits C_1C_2, as spelled out at page 2, second paragraph of the SUMMARY. Similarly, so is the second sentence of the SUMMARY.

Appropriate correction is required.

2. Claims 1-15 are rejected under 35 U.S.C. 112, first paragraph, as based on a disclosure which is not enabling. The "conversion of existing symbolic date representations ... without the addition of new data fields", as indicated at page 2 lines 7-10, is critical or essential to the practice of the invention, but not included in the claim(s) is not enabled by the disclosure. *In re Mayhew*, 527 F.2d 1229, 188 USPQ 356 (CCPA 1976).

The problem set forth in the last four lines of page 1 and promised in the first paragraph of page 2, as well as in the lines quoted above, indicate that the invention solves the Y2K problem without introducing additional digits. The claims, the abstract, and the description of the invention in the SUMMARY clearly involve century digits C_1C_2, which increase the number of date digits from 6 to 8, thus using 4 digits to indicate the year. One of ordinary skill in the art would not know how to resolve this discrepancy.

3. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless --
>
> (a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

Claims 1-15 are rejected under 35 U.S.C. 102(a) as being clearly anticipated by The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation, Third Edition, May, 1996, IBM Corp.

Remark: The claims are clearly anticipated by the windowing techniques, discussed in some detail beginning at page 4-3. Note in particular the use of 19 and 20 for century digits at page 4-3. Sorting is addressed at 7-23 under DFSORT.

4. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

5. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Wayne Amsbury whose telephone number is (703) 305-3828. The examiner can normally be reached on Monday-Thursday from 6:30 AM to 5:00 PM Eastern time.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas G. Black, can be reached on (703) 305-9707. The fax phone number for this Art Unit is (703) 305-9731.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-9600.

November 6, 1997

Copy 2

CLAIMS 1-15 ARE PENDING

1. The disclosure is objected to because of the following informalities:

The statement of the problem, at the bottom of page 1, which implies that requiring additional data fields for storage to solve the Y2K problem, is contradicted by the use of century digits C_1C_2, as spelled out at page 2, second paragraph of the SUMMARY. Similarly, so is the second sentence of the SUMMARY.

Appropriate correction is required.

2. Claims 1-15 are rejected under 35 U.S.C. 112, first paragraph, as based on a disclosure which is not enabling. The "conversion of existing symbolic date representations ... without the addition of new data fields", as indicated at page 2 lines 7-10, is critical or essential to the practice of the invention, but not included in the claim(s) is not enabled by the disclosure. *In re Mayhew*, 527 F.2d 1229, 188 USPQ 356 (CCPA 1976).

The problem set forth in the last four lines of page 1 and promised in the first paragraph of page 2, as well as in the lines quoted above, indicate that the invention solves the Y2K problem without introducing additional digits. The claims, the abstract, and the description of the invention in the SUMMARY clearly involve century digits C_1C_2, which increase the number of date digits from 6 to 8, thus using 4 digits to indicate the year. One of ordinary skill in the art would not know how to resolve this discrepancy.

3. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless --
>
> (a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

Claims 1-15 are rejected under 35 U.S.C. 102(a) as being clearly anticipated by The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation, Third Edition, May, 1996, IBM Corp.

Remark: The claims are clearly anticipated by the windowing techniques, discussed in some detail beginning at page 4-3. Note in particular the use of 19 and 20 for century digits at page 4-3. Sorting is addressed at 7-23 under DFSORT.

4. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

5. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Wayne Amsbury whose telephone number is (703) 305-3828. The examiner can normally be reached on Monday-Thursday from 6:30 AM to 5:00 PM Eastern time.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas G. Black, can be reached on (703) 305-9707. The fax phone number for this Art Unit is (703) 305-9731.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-9600.

November 6, 1997

#7

OIPE
MAR 20 1998
JC21
PATENT & TRADEMARK OFFICE

Attorney's Docket No. 08190-0119.000 PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Bruce Dickens
Serial No.: 08/725,574
Filed: October 3, 1996
For: DATE FORMATTING AND SORTING FOR SPANNING THE TURN OF THE CENTURY

Group Art Unit: 2307
Examiner: W. Amsbury

March 17, 1998

Assistant Commissioner for Patents
Washington, DC 20231

RESPONSE

Sir:

This Response is filed to address the issues raised by the Office Action dated November 17, 1997. As described below, Applicant submits that the disclosure is enabling and that both the disclosure and claims comply with 35 U.S.C. § 112, first paragraph. Further, the Applicant submits that the claimed invention was conceived and reduced to practice before the publication date of the cited IBM article. Thus, Applicant submits that the rejection of the claims under 35 U.S.C. § 102(a) is thereby overcome.

I. The Invention

As stated in the application, many existing databases contain date representations that are defined only by the decade and year designations (i.e., Y_1Y_2). Because these databases do not provide date designations for the century associated with each date, it will be impossible to discern the order of dates in a database after the turn of the century.

To properly and efficiently address this problem, a method for converting dates in databases was needed. This method should accept the dates from data storage, discern the

proper century designation for each date, and reformat the dates with the century designation.

The claimed invention provides a method of processing symbolic representations of dates stored in a database. The method of the claimed invention includes the step of providing a database that contains dates represented in symbolic representations in the form of $M_1M_2D_1D_2Y_1Y_2$, where M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator. The method of the claimed invention then selects a ten-decade window, beginning with the decade designated by Y_AY_B. Not only does Y_AY_B identify the first decade in the ten-decade period, but Y_AY_B is earlier than the earliest Y_1Y_2 year designator in the database (i.e., the earliest date in the database). The method of the claimed invention further includes the step of determining a century designator C_1C_2 for each symbolic representation of a date in the database based on the previously selected ten-decade period. In particular, the determining step will determine a first value for C_1C_2 if the Y_1Y_2 date representation for the date is less than Y_AY_B. On the other hand, the determining step will determine a second value for C_1C_2 if Y_1Y_2 is greater than Y_AY_B. Finally, the method of the claimed invention includes the step of reformatting the symbolic representations of the dates into corresponding values of C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2. These values can then be used to manipulate the dates, such as by sorting the dates in chronological order.

In a typical operation of the claimed invention, the method performs date conversions on a database that includes dates from both the twentieth and twenty-first century. In this operation, the method of the claimed invention initially selects a ten-decade window where the first year of the first decade is earlier than the earliest date in the database. For example, if the earliest date in the database is 1961, the

Response #7

selecting step of the present method may choose a ten-decade period of 1960 - 2059.

After the selecting step has determined a proper ten-decade window, the method of the claimed invention determines a century designation C_1C_2 for each of the dates in the database. For example, if two date representations in the data base are 01/01/75 and 01/01/49, the determining step of the present invention would determine that 01/01/75 is 01/01/1975 because 75 > 60. Further, the determining step of the present invention would determine that 01/01/49 as 01/01/2049 because 49 < 60. The method of the present invention further includes the step of reformatting the symbolic representations of the dates into values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2. In other words, the date 01/01/75 is reformatted as 19570101 and the date 01/01/49 is reformatted as 20490101. These dates can then be used for several operations such as date sorting.

Advantageously, the method of the claimed invention can be implemented as an initial step in any database manipulation program. For instance, the method of the claimed invention may be embodied in computer software code that preprocesses a database prior to beginning the remainder of the data manipulation program. In this embodiment, the method initially converts the data from the varying formats, determines a century designation for each date, and reformats the dates such that the dates may be used by the database manipulation program for such operations as sorting and printing the dates. In this embodiment of the present invention, the dates are temporarily converted and reformatted for use by the manipulation program. However, the method of the present invention need not store the converted date in data storage. Instead, the original dates in data storage remain undisturbed. This aspect of the present invention thus allows conversion of dates to compensate for century

designations without requiring the addition of data fields to permanently store the century designations.

II. The Specification and Claims Comply with 35 U.S.C. § 112

In paragraph 1, the Office Action objects to the disclosure for implying that the current invention does not require additional data fields for storage to solve the year 2000 problem. Further, in paragraph 2, the Office Action rejected all of the claims under 35 U.S.C. § 112, first paragraph, as based on a disclosure that is not enabling. In particular, the Office Action states that the conversion of the existing symbolic date representations without the addition of new data fields is critical or essential to the invention but is not included in the claims.

As described below, the method of the claimed invention does not require that the converted data that includes the century designations be stored in data storage. Likewise, the Applicant respectfully submits that the amended set of claims does not require storage of the converted dates and therefore imposes no requirement for new data fields. Thus, Applicant respectfully submits that the disclosure is in accordance with 35 U.S.C. § 112 and is thus enabling.

As stated in the background of the invention, conventional date formatting systems typically require additional data fields for storage to accommodate the century designators. These additional data fields are necessary because conventional systems disclose a permanent reformatting of the stored data. The claimed invention, on the other hand, does not require that the reformatted data be permanently stored. Instead, the method of claimed invention encompasses embodiments in which the date information is initially reformatted and converted to have century designations, but does not require that the reformatted dates be stored. As stated previously, the method of one embodiment of the claimed

invention reads the dates from the database and temporarily reformats the dates with century designations. Data manipulation programs are then performed on these reformatted dates, such as sorting the dates. However, once the data manipulations are complete, the reformatted dates need not be stored in data storage. Instead, the dates in the data storage can remain the same as they were prior to the temporary reformatting of the data by the method of the claimed invention. Thus, in these embodiments, the method of the claimed invention does not require additional data fields for storage because the reformatted dates with century designations are only used "on the fly" for data manipulation and are not stored in data storage.

For the reasons described above, Applicant therefore submits that the disclosure is enabling and that both the disclosure and the claims comply with 35 U.S.C. § 112, first paragraph. As such, the rejections under 35 U.S.C. § 112 are therefore overcome.

III. The Claimed Invention was Reduced to Practice Prior to the Publication Date of the Cited Reference.

In paragraph 3, the Office Action rejected all of the claims under 35 U.S.C. § 102(a) as being anticipated by an IBM article entitled *The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation* (3d. ed. May 1996). Applicant submits herewith a Declaration under 37 C.F.R. § 1.131, signed by Bruce Dickens, the inventor of the above-identified application (hereinafter the "Dickens Declaration"). As stated in the Dickens Declaration, the method of the present invention was conceived and reduced to practice at least as early as April 4, 1996. (See Dickens Declaration, page 4, paragraph 9). In this regard, a computer program dated April 4, 1996 embodying one advantageous embodiment of the claimed invention was created at least as early as April 4, 1996.

This program was submitted as an Exhibit at the time of filing the present application and is attached as Exhibit G to the Dickens Declaration. As also set forth in paragraph 7 of the Dickens Declaration and as evident from the face of the computer program attached as an Exhibit to the present application and as Exhibit G to the Dickens Declaration, the April 4, 1996 date of the program predates the May 1996 publication date of the IBM article. Since the claimed invention was conceived and reduced to practice before the publication date of the cited IBM article, the third edition of the IBM article is therefore removed as a reference for purposes of 35 U.S.C. § 102(a). Thus, Applicant submits that the rejection of the claims under 35 U.S.C. § 102(a) is also thereby overcome.

In addition to the Declaration, the Applicant also submits herewith a Supplemental Information Disclosure Statement that discloses the first edition of the IBM article date October 1995. However, as stated in the Declaration, the Applicant conceived of the method of the claimed invention at least as early as February 1995. (See Dickens Declaration, page 2, paragraph 4). Further, the Applicant diligently worked to reduce to practice the claimed method from a date prior to October 1995 to its reduction to practice at least as early as April 4, 1996 as evidenced by the above-referenced computer program attached as Exhibit G to the Dickens Declaration. (See also Dickens Declaration, page 4, paragraph 9).

CONCLUSION

In view of the Declaration and the remarks presented above, it is respectively submitted that all of the present claims of the application are in condition for immediate allowance. It is therefore respectfully requested that a Notice of Allowance be issued. The Examiner is encouraged to

contact Applicant's undersigned attorney to resolve any remaining issues in order to expedite examination of the present application.

It is not believed that extensions of time are required, beyond those which may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any additional fee required therefore (including fees for net addition of claims) is hereby authorized to be charged to Deposit Account No. 16-0605.

Respectfully submitted,

Guy R. Gosnell
Registration No. 34,610

BELL SELTZER INTELLECTUAL PROPERTY LAW GROUP
ALSTON & BIRD LLP
Post Office Drawer 34009
Charlotte, NC 28234 — Atlanta
Tel (704) 331-6000
Fax (704) 334-2014

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Assistant Commissioner of Patents, Washington, DC 20231, on March 17, 1998.

W. Kevin Ransom

314064

OFFICIAL

PATENT

Attorney's Docket No. 08190-0119.000

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Bruce Dickens
Serial No.: 08/725,574
Filed: October 3, 1996
For: DATE FORMATTING AND SORTING FOR SPANNING THE TURN OF THE CENTURY

Group Art Unit: 2307
Examiner: W. Amsbury

April 2, 1998

Assistant Commissioner for Patents
Washington, DC 20231

SUPPLEMENTAL RESPONSE

Sir:

This Supplemental Response is intended to supplement the Response filed March 17, 1998 by amending the above-identified patent application as follows:

In The Claims:

Please amend independent Claims 1, 4, 6, 8, 10, 11 and 13 as follows:

1. (Amended) A method of processing symbolic representations of dates stored in a database, comprising the steps of

providing a database with symbolic representations of dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later

than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each symbolic representation of a date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B; and

reformatting the symbolic representation of the date [in the database] with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2 <u>to facilitate further processing of the dates</u>.

Claim 4, line 3, please delete "in the database".

Claim 6, line 3, please delete "in the database".

Claim 8, line 1, please delete "1", and insert -- 10 -- therefor.

Claim 10, line 3, please delete "sorted".

11. (Amended) A method of processing dates in a database, comprising the steps of

providing a database with dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of dates falling within a 10-decade period of

time which includes the decade beginning in the year 2000;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B; [and]

reformatting each date [in the database] in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$ to facilitate further processing of the dates; and

sorting the dates [in the database] in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$.

Claim 13, line 1, please delete "11", and insert -- 15 -- therefor.

REMARKS

As noted by the prior Response filed March 17, 1998, the first Official Action objected to the disclosure for implying that the current invention does not require additional data fields for storage to solve the year 2000 problem. The Official Action also rejected all of the claims under 35 U.S.C. § 112, first paragraph, as based on a disclosure that is not enabling. In particular, the Official Action stated that the conversion of the existing symbolic

Supplemental Response

date representations without the addition of new data fields is critical or essential to the invention, but is not included in the claims.

As stated in the Background Of The Invention section of the present application, conventional date formatting systems typically require additional data fields for storage to accommodate the century designators. These additional data fields are necessary because conventional systems permanently reformat the stored data. In contrast, the method of the present invention, as described by the specification, does not require that the converted or reformatted data that includes century designations be stored in data storage. Accordingly, independent Claims 1 and 11 have now been amended so as to not require storage of the converted dates, thereby not imposing any requirement for new data fields.

Thus, the method of the claimed invention encompasses embodiments in which the date information is initially reformatted and converted to have century designations, but does not require that the reformatted dates be stored. For example, the method of one embodiment of the claimed invention reads the dates from the database and temporarily reformats the dates with century designations. Data manipulation programs are then performed on these reformatted dates, such as sorting the dates. However, once the data manipulations are complete, the reformatted dates need not be stored in data storage. Instead,

the dates in data storage can remain the same as they were prior to the temporary reformatting of the date information. Thus, the method of this embodiment of the claimed invention does not require additional data fields for storage because the reformatted dates with century designations are only used "on the fly" for data manipulation and are not stored in data storage.

In view of the amendments to the independent claims, dependent Claims 4 and 6 have also been amended to no longer refer to symbolic representations of dates "in the database". Since dependent Claims 10 and 15 further recite storing the reformatted dates in the database, dependent Claims 8 and 13 which further describe manipulating information in the database that includes the reformatted dates have also been amended to depend from dependent Claims 10 and 15, respectively. Finally, dependent Claim 10 has been amended to delete reference to "sorted" symbolic representations of the dates to provide proper antecedent basis.

For the reasons described above, Applicant therefore submits that the disclosure is enabling and that both the disclosure and the claims, as amended, comply with 35 U.S.C. § 112, first paragraph. As such, the rejections raised by the first Official Action under 35 U.S.C. § 112 are therefore overcome.

In re: Bruce Dickens
Serial No.: 08/725,574
Filed: October 3, 1996
Page 6

CONCLUSION

In view of the amended claims and the remarks presented above, it is respectively submitted that all of the present claims of the application are in condition for immediate allowance. It is therefore respectfully requested that a Notice of Allowance be issued. The Examiner is encouraged to contact Applicant's undersigned attorney to resolve any remaining issues in order to expedite examination of the present application. The Commissioner is also hereby authorized to charge our Deposit Account No. 16-0605 for any additional extensions of time that are required for entry and consideration of this Supplemental Response beyond the one month extension of time that was previously obtained in conjunction with the prior Response.

Respectfully submitted,

Guy R. Gosnell
Registration No. 34,610

~~BELL SELTZER INTELLECTUAL PROPERTY LAW GROUP~~
ALSTON & BIRD LLP
Post Office Drawer 34009
Charlotte, NC 28234
Tel (704) 331-6000
Fax (704) 334-2014

CERTIFICATION OF FACSIMILE TRANSMISSION

I hereby certify that this paper is being facsimile transmitted to the Patent and Trademark Office on the date shown below.

Gwen Frickhoeffer
Gwen Frickhoeffer

April 2, 1998
Date

326124

Chronology of Dickens Patent (5,806,063) prosecution

10/3/96 **Filing Date of the application.**

11/17/97 **Mail Date of the first office action.**

Rejections in case:

1. 35 U.S.C. 102(e) rejection under "The Year 200 and 2-Digit Dates: A Guide for Planning and Implementation, Third Edition, May 1996."
2. 35 U.S.C. 112, first paragraph rejection. Examiner alleges that the claims increase the number of date digits, which is inconsistent with the specification, which indicate that the invention "solves the Y2K problem without introducing additional digits."

3/20/98 **Response to first office action filed.**

Key point in response: "dates are temporarily converted and reformatted...Instead, the original dates in data storage remain undisturbed...This aspect of the invention thus allows conversion of dates to compensate for century designations without requiring the addition of data fields to permanently store the century designations." Applicant overcomes 35 U.S.C 112 by explanation of invention.

Second Key point: Applicant swears behind May 1996 reference with a Declaration under 37 C.F.R. 131.

4/2/98 **Telephone interview conducted.**

Key point in the interview: Examiner indicated possible allowance of case if claim was amended to indicate that "the use of additional century digits did not include their storage in the database."

4/2/98 **Supplemental response filed.**

Key point in response: "Claims 1 and 11 have now been amended so as to not require storage of the converted dates... the claimed invention does not require additional data fields for storage."

4/8/98 **Notice of allowance mailed.**

Key point: Examiner's reason for allowance stresses two main points, namely, that the prior art does not disclose "threshold year digits as used to determine a pair of century digits to be used for computation, but without enlarging the number date digits of the database."

EXAMINER'S ANALYSIS OF USPAT 5,806,063 CLAIM 1

A method of processing symbolic representations of dates stored in a **database**, comprising the steps of:	There are a variety of Y2K problems, of which one is to use a database that contains 2-digit date fields. These must be **recognized**, not a trivial problem in some cases. This differs from **correction** of a database by either modification or creation of a new database, and both problems are related to but distinct from dealing with the software rather than the database.
providing a database with symbolic representations of dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the **symbolic representations** of dates falling within a 10-decade period of time;	The parameters, such as M_1M_2, of which there are 5 pairs in the claim, must be addressed in some fashion in any valid reference. Many of the general discussions fail to do so explicitly. For instance, a potential reference may simply interpret the 2-digit years in a range as inherently being 20th century years, without forming a symbolic representation that includes C_1C_2 as indicated below.
selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the **earliest Y_1Y_2 year designator** in the database;	Selecting here is interpreted as **searching** for the earliest 2-digit year in the database, in light of col 1 lines 51-56 of the SUMMARY, where the invention is applied in a manner which "requires no user input".
determining a **century designator C_1C_2** for each symbolic representation of a date in the database, C_1C_2 **having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B**; and	The application of C_1C_2 is important because it allows the database dates to be sorted properly. The use of Y_AY_B determined by the search for the earliest date **differs significantly from the use of the system clock to determine this value, as is done in LISP and proposed in various references**.
reformatting the symbolic representation of the date with the values C_1C_2, Y_1Y_2, M_1M_2 and D_1D_2 **to facilitate further processing of the dates**.	Further processing includes **sorting** with the century digits **in the leading position**, as noted at col 2, see col 2 lines 16-21.

PALM INTRANET

Day : Wednesday
Date: 12/15/1999
Time: 07:41:49

Content Information for 08/725574

Search Another: Serial# Search **or Patent#** Search

Serial Info **Contents** Attorney/Agent Info Continuity Data Foreign Data Inventors

Num	Type	Date	Code	Contents Description
30	Y	09/09/1999	FOND	CASE FOUND
29	Y	09/02/1999	LOST	CASE WAS REPORTED LOST
28	C	11/30/1998	N423	OCERTIFICATES OF CORRECTION (PTO-1050) P/E
27	O	09/08/1998	PGM/	PATENT GRANT MAILED
26	O	08/03/1998	WPIR	WEEKLY PATENT ISSUE RECEIPT
25	O	06/24/1998	DGP1	DDRAWINGS PROCESS COMPLETED FOR DRAWING SET NUMBER 1.
24	I	06/24/1998	DGM1	DDRAWINGS MATCHED TO APPLICATION, 1=DRAWING SET NUMBER.
23	I	05/30/1998	DGCR	APPL RECEIVED IN DRAWING PROCESS DIV FOR FILING OF DRAWINGS.
22	B	05/22/1998	N084	BASE ISSUE FEE PAYMENT
21	O	05/14/1998	DGCO	APPL ORDERED FROM DRAWING PROCESS DIV FOR FILING OF DRAWINGS
20	I	05/14/1998	DGU1	DDRAWINGS RECEIVED IN DPD, 1=DRAWING SET NUMBER
19	I	05/11/1998	DGR1	MMAIL ROOM DATE OF DRAWING, 1=DRAWING SET NUMBER.
18	N	04/08/1998	N/=.	NOTICE OF ALLOWANCE PRINT
17	A	04/03/1998	CNTA	COUNT DATE-NOTICE OF ALLOWANCE; IF TYPE F-1ST ACTION; IF TYPE M-2ND/PLUS ACTION FAOM; IF TYPE A-ALL OTHER ACTIONS
16	E	04/03/1998	FWDX	DATE FORWARDED TO EXAMINER
15	I	04/02/1998	SA..	SUPPLEMENTAL RESPONSE
14	O	04/02/1998	EXIN	EXAMINER INTERVIEW SUMMARY RECORD

13	I	03/20/1998	M844	PRIOR ART CITATION FILED P/E
12	E	04/01/1998	FWDX	DATE FORWARDED TO EXAMINER
11	I	03/20/1998	A...	RESPONSE AFTER NON-FINAL ACTION
10	I	03/20/1998	XT/G	FIRST REQUEST FOR EXTENSION OF TIME GRANTED (INCLUDING TO FILE BRIEF)
9	O	11/17/1997	MAIL	MAIL DATE OF OFFICE ACTION
8	F	11/07/1997	CTNF	COUNT DATE-NON-FINAL ACTION; IF TYPE F-1ST ACTION; IF TYPE M-2ND/PLUS ACTION FAOM; IF TYPE O-ALL OTHER ACTIONS
7	O	10/23/1997	N570	COMMUNICATION RE POWER OF ATTORNEY (PTOL-308, 46-90) P/E
6	I	01/31/1997	PA..	CHANGE IN POWER OF ATTORNEY (MAY INCLUDE CORR. CHANGE) (MAY INCLUDE ASSOC. POWER) P/E
5	I	04/16/1997	M844	PRIOR ART CITATION FILED P/E
4	D	10/16/1997	DOCK	DATE CASE WAS DOCKETED
3	E	08/07/1997	TR.Q	TRANSFER INQUIRY
2	I	12/16/1996	FILM	MICROFILM RECORD CAPTURED OF APPLICATION
1	E	10/17/1996	IE05	INITIAL EXAM TEAM 5

Serial Info | **Contents** | Attorney/Agent Info | Continuity Data | Foreign Data | Inventors

(To Go BACK Use BACK Button on Your BROWSER Tool Bar)

Back to || PALM || ASSIGNMENT || OASIS || Home Page

REEXAMINATION CONTROL NUMBER__________ GROUP ART UNIT____________

ORDERED BY____

TELEPHONE NUMBER ___________ DATE OF ORDER____________

ORDER FORM FOR REEXAM REFERENCES

TO: OFFICE OF PUBLIC SERVICES—PROGRAM CONTROL DIVISION
SUBJECT: **REQUEST FOR EXPEDITED SERVICE**

PLEASE PROVIDE ONE COPY OF THE FOLLOWING US PATENTS. DELIVER THEM TO THE PUBLIC SEARCHROOM "PUBLIC SERVICE WINDOW"

PATENT NUMBER	PATENT NUMBER
____________	____________
____________	____________
____________	____________
____________	____________

GROUP INSTRUCTIONS: (1) LIST US PATENTS PRINTED ON PATENT UNDERGOING REEXAMINATION IN ASCENDING NUMERICAL ORDER.
(2) DELIVER ORDER FORM TO THE PUBLIC SERVICE WINDOW (CP4—LOBBY)

PUBLIC SERVICE WINDOW: UPON RECEIPT OF THE COMPLETED ORDER, TELEPHONE THE PERSON REQUESTING THE COPIES THAT THE COPIES ARE READY.

FORM PTO-892 (REV. 3-78)

U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

CONTROL NO.	GROUP ART UNIT	ATTACHMENT TO PAPER NUMBER	
90/005,592	2771		1

NOTICE OF REFERENCES CITED

PATENTEE

Bruce Dickens

U.S. PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
	A	5630118	5/13/97	Shaughnessy	707	1	11/21/94
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

FOREIGN PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUB-CLASS	PERTINENT SHTS. DWG	PP. SPEC.
	L								
	M								
	N								
	O								
	P								
	Q								

OTHER REFERENCES (Including Author, Title, Date, Pertinent Pages, Etc.)

R Ohms, Computer Processing of dates outside the twentieth century, IBM Systems Journal, Vol. 25, No. 2, 1986

S

T	
U	

EXAMINER	DATE	
Amsbury	12/21/99	

* A copy of this reference is not being furnished with this office action.
(See Manual of Patent Examining Procedure, section 707.05 (a).)

15

FORM PTO-892 (REV. 3-78)

U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

NOTICE OF REFERENCES CITED

CONTROL NO.	GROUP ART UNIT	ATTACHMENT TO PAPER NUMBER
90/005,592	2771	1

PATENTEE: Bruce Dickens

U.S. PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
	A	5630118	5/13/97	Shaughnessy	707	1	11/21/94
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

FOREIGN PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUB-CLASS	PERTINENT SHTS. DWG	PERTINENT PP. SPEC.
	L								
	M								
	N								
	O								
	P								
	Q								

OTHER REFERENCES (Including Author, Title, Date, Pertinent Pages, Etc.)

R	Ohms, Computer Processing of dates outside the twentieth century, IBM Systems Journal, Vol. 25, No. 3, 1986
S	
T	
U	

EXAMINER	DATE
Amsbury	12/21/99

* A copy of this reference is not being furnished with this office action.
(See Manual of Patent Examining Procedure, section 707.05 (a).)

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Tra[illegible]ark Office
Address: ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

CONTROL NUMBER	ORDER DATE	PATENT NUMBER	PATENTEE
90/005,592	12-21-99	5,806,063	Dickens

GUY R. GOSNELL
BELL, SELTZER, PARK & GIBSON, P.A.
P.O. DRAWER 34009
CHARLOTTE , NC 28234

EXAMINER	
Thomas Black	
ART UNIT	PAPER NUMBER
2771	1

DATE MAILED: December 21, 1999

COMMISSIONER INITIATED ORDER FOR REEXAMINATION

Attachment(s): ☒ PTO-892. ☐ PTO-1449.
☐ Other: ____________

Response Time For Patent Owner's Statement:

TWO MONTHS from the date hereof. 37 CFR 1.530(b).

Notes: If the patent owner does not file a timely statement under 37 CFR 1.530(b), reexamination will proceed in accordance with 37 CFR 1.550(a).

An identification of the claims, the references relied on, and the rationale the decision to order reexamination is attached.

REEXAMINATION ORDER:

Pursuant to 37 CFR 1.520, reexamination is ordered. Note the attached decision.

DECISION

Pursuant to 37 CFR § 1.520, the Commissioner of Patents and Trademarks has determined that the prior art discussed below raises a substantial new question of patentability as to claims 1-15 of U.S. Patent No. 5,806,063 .

Claims 1-15 of U.S. patent No. 5,806,063, as a whole, are drawn to a method particularly useful for extending processing dates stored in a database beyond the turn of the century. The method comprises the steps of providing a database with dates stored in it according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the dates falling within a 10 decade period of time. A 10 decade window with a Y_AY_B value for the first year of the 10 decade window is selected, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database. A century designator C_1C_2 is determined for each date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B. Each date in the database is assigned the values C_1C_2 ,Y_1Y_2 ,M_1M_2, and D_1D_2 . In one case that produces particularly advantageous results for many operations, such as chronological date sorting, the date can then be represented in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$.

The '063 patent points out that, for the case of most practical interest, the 10 decade period of time (window) spans the year 2000 (i.e., includes decades earlier than the year 2000 and decades later than the year 2000), and it begins with a year in which the second digit of Y_AY_B is 0 (zero). Further, assume that a decade beginning in 1950 is selected as the first decade of the 10-decade window, i.e., Y_A is selected as 5. Then, C_1C_2 is assigned "20" if Y_1 is less than 5 and is assigned "19" if Y_1 is equal to or greater than 5. In that case, if Y_1Y_2 is "43", the century designator C_1C_2 is "20," indicating that the year in question in the database is 2043. On the other hand, if Y_1Y_2 is "63", the century designator C_1C_2 is "19," indicating that the year in question in the database is 1963. For chronological date sorting, the date is represented in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$; thus, the date 12/15/93 (Dec. 15, 1993) is represented as 19931215 and the date 12/15/00 (Dec. 15, 2000) as 20001215.

The Ohms Article:

The article B. G. Ohms, Computer Processing of Dates Outside the Twentieth Century, IBM Systems Journal, Vol. 25, No. 2, 1986 is relevant to claims 1-15 of the '063 patent. As background to explaining the relevance of the Ohms article, the broadest reasonable interpretation of the claims is set forth as follows:

There are eight parameters listed in claim 1, in the form of four pairs. In the absence of any separation of the elements of the pairs, as for instance Y_1 from Y_2 in Y_1Y_2, the pair is taken to designate a two-digit single value. Thus $Y_1Y_2 = 43$, not $Y_1 = 4$ and $Y_2 = 3$, and so, this notation is equivalent to a year designation as YY, which is inherently ordered as Y_1Y_2 by the positional nature of the decimal system.

In the absence of the use of individual decades, a 10-decade window is taken to be equivalent to a century window. Similarly, the use of Y_B as zero in some of the claims is taken as an arbitrary choice because there is no claim of use of this particular value, as there would be if the fast-sort techniques noted in the specification were claimed.

In light of column 3, lines 35-38, the selection of Y_AY_B in the claim is taken to be external to the part of the method which is automated. The determination of the earliest year date is taken to include the possibility that the user may have prior information about this number when Y_AY_B is chosen, and thus it is not required to include a search for this number within the method.

The reformatting of claim 1 is not limited to the specific one recited in claim 5.

Furthermore, the '063 patent claims can be reasonably interpreted such that the date formats that include the century digits C_1C_2 need not be stored into the database; the reformatting can be dynamic and created for the purpose of processing dates extracted from the database. Thus, in claim 1, for instance, the century designator is stated (in the claim) to be "for each symbolic representation of the date in the database;" however, it is not considered to be part of the symbolic representation but is related to it. The storage of "associated information" as in claim 9 is not required to be the century designation C_1C_2, and is not to occupy the date field(s).

Thus, the Ohms article is relevant as follows:

<u>As to independent claim 1 of the '063 patent:</u>

> **"A method of processing symbolic representations of dates stored in a database, comprising the steps of:"**

See the Ohms abstract at page 244, left hand column.

> **"Providing a database with symbolic representation of dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;"**

The eight parameters of the claim are not explicit in Ohms, which uses simply YYMMDD for the short Gregorian format, table 1 at page 247. As noted above, the YYMMDD of Ohms is equivalent to the $Y_1Y_2M_1M_2D_1D_2$ of the claim because the claim does not require a separate use of the digits.

> **"Selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;**

See Ohms page 248, right hand column, lines 10-18, where a year is specified as "the starting point of a 100-year range." In many cases, the range of dates in a database is known, and dates of only one century occur in the database. In others, the effective four year range of (2-digit) dates in the 20th century is known. In both cases, Y_AY_B can be selected to separate the dates of one century from another by choosing it to be less than or equal to Y_1Y_2.

The skilled artisan at the time of the invention would have been motivated to select the pivot Y_AY_B less than or equal to Y_1Y_2 because it would separate 2-digit dates into the proper century in an efficient manner by simply comparing them to Y_AY_B.

> **"Determining a century designator C_1C_2 for each symbolic representation of a date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B."**

See Ohms page 248, right hand column, lines 10-18, where the effect of the pivot 25 determines 1925 while it also determines 2024, (from the dates with year digits of 25 and 24).

The distinction between the earliest Y_1Y_2 and Y_AY_B allows for the use of convenient separators, such as the 80-year window noted at page 248, right hand column, lines 28-38. The skilled artisan at the time of the invention would have been motivated to provide that Y_AY_B not be equal to the earliest Y_1Y_2 because it permits a convenient way of segregating the data between two centuries.

<u>**As to claims 2-3 of the '063 patent:**</u>

The examples of Ohms include the decade beginning in the year 2000, and the C_1C_2 values of 19 and 20.

<u>**As to claim 4 of the '063 patent:**</u>

Although the sorting of the symbolic values is not specifically addressed in Ohms; it would have been readily apparent to sort the date values, because this makes it possible to efficiently prioritize records that include dates, such as orders to be filled.

<u>**As to claims 5-6 of the '063 patent:**</u>

Ohms converts to a Julian date with leading 4-digit year from a 2-digit year format. This places the century first and makes sorting more efficient, and in particular places the 20th century dates before the 21st century dates, as they should be. This conversion would be equally applied to the Gregorian dates for the same reason Ohms applies it to the Julian dates.

<u>**As to claim 7 of the '063 patent:**</u>

The skilled artisan would have been motivated to preconvert databases not in the format required for the method of claim 1 in order to avoid the need for development of a separate system.

As to claim 8 of the '063 patent:

Ohms does not specify a value of $Y_B = 0$; however, the use of a particular second digit Y_B has no criticality in the claimed invention, - it is merely one of the 10 possible choices. Absent a showing of criticality, this would have been well within knowledge and level of skill of the ordinary artisan.

As to claim 9 of the '063 patent:

As to the claim 9 storage, see Ohms, page 248, right hand column, near the bottom, where it states with respect to storage "Of course, in the vast majority of cases, that is exactly what does take place."

As to claim 10 of the '063 patent:

The skilled artisan would have been motivated to manipulate the information in the database, because the point of the conversion is to provide for manipulating the information to avoid a Y2K problem.

Claims 11-15:

Claim 11 corresponds to claims 4-5 combined, claim 12 to claims 1+ 4+7, claim 13 to claims 1+4+8, claim 14 to 1+4+9, claim 15 to 1+4+10. Thus, Ohms is relevant to the elements of claims 11-15 accordingly, based on the analysis set forth above.

The Shaughnessy patent:

Shaughnessy patent 5,630,118 is generally directed to software patches of application programs, and in particular to providing a basis for proper subtraction of dates. [Abstract] The patent determines date pivots within a century span by using the current date [FIG 2,4], and install date [FIG 4], and sometimes an offset from these [FIG 4, numeral **18**]. It describes a variety of date formats including that of CCYYMMDD, which is related to the more parameterized $C_1C_2Y_1Y_2M_1M_2D_1D_2$ date format of the '063 patent. The preferred conversion is to a YYYYMMDD format. There is a fairly broad discussion of date windowing within these constraints.

Shaughnessy addresses the use of the earliest date in a **database**, through an implicit assumption that this is related to the install date of an application program, and the (unspoken) possibility that such a program might be a data base management system (DBMS). It does allow for a fixed arbitrary date to be entered [col 6 lines 30-35], but the context of this is a database in which there are dates both above and below the chosen pivot date.

Conclusion:

In view of the above teachings, a substantial new question of patentability is raised as to claims 1-15 of U.S. Patent No. 5,806,063. Reexamination of U.S. Patent No. 5,806,063 is hereby ordered under 37 CFR § 1.520. All the patent claims will be reexamined.

Q. Todd Dickinson
Assistant Secretary of Commerce and
Commissioner of Patents and Trademarks

kenccomr\5806063.cmr

US005806063A

United States Patent [19]

Dickens

[11] Patent Number: 5,806,063

[45] Date of Patent: Sep. 8, 1998

[54] DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

[75] Inventor: Bruce Dickens, Irvine, Calif.

[73] Assignee: McDonnell Douglas Corporation, Long Beach, Calif.

[21] Appl. No.: 725,574

[22] Filed: Oct. 3, 1996

[51] Int. Cl.[6] G06F 17/30

[52] U.S. Cl. 707/6; 707/102; 707/7; 707/200

[58] Field of Search 707/6, 102, 7, 707/200

[56] References Cited

U.S. PATENT DOCUMENTS

4,573,127	2/1986	Korff	364/493
5,630,118	5/1997	Shaughnessy	707/1
5,644,762	7/1997	Soeder	707/6
5,668,989	9/1997	Mao	707/101

OTHER PUBLICATIONS

The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation, Third Edition, May, 1996.

IBM: *The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation*; First Edition, Oct. 1995.

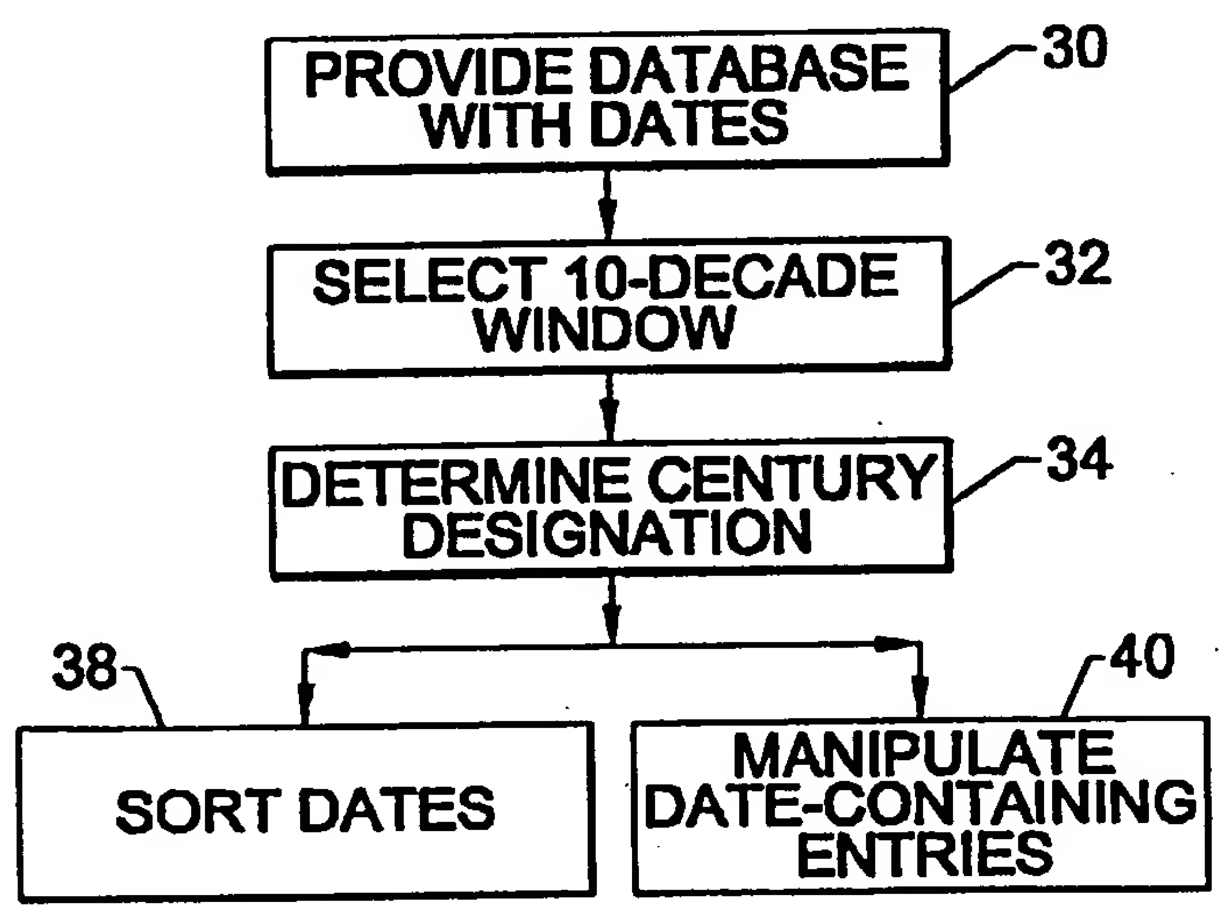
Primary Examiner—Wayne Amsbury

Attorney, Agent, or Firm—Bell Seltzer Intellectual Property Group of Alston & Bird LLP

[57] ABSTRACT

Dates stored in symbolic form in a database are reformatted to permit easy manipulation and sorting of date-related information. Each date in M_1M_2, D_1D_2, and Y_1Y_2 format is converted to C_1C_2, Y_1Y_2, $_{M1}M_2$, and D_1D_2 format. To accomplish the conversion, a 10-decade window starting on Y_AY_B is defined that encompasses all dates in the database. The value of C_1C_2 is determined by the relative values of Y_1Y_2 and Y_AY_B. The reformatted date information is particularly useful when the reformatting is in $C_1C_2Y_1Y_2M_1M_2D_1D_2$ format, because sorting by date is accomplished using a pure numerical-value sort.

15 Claims, 1 Drawing Sheet



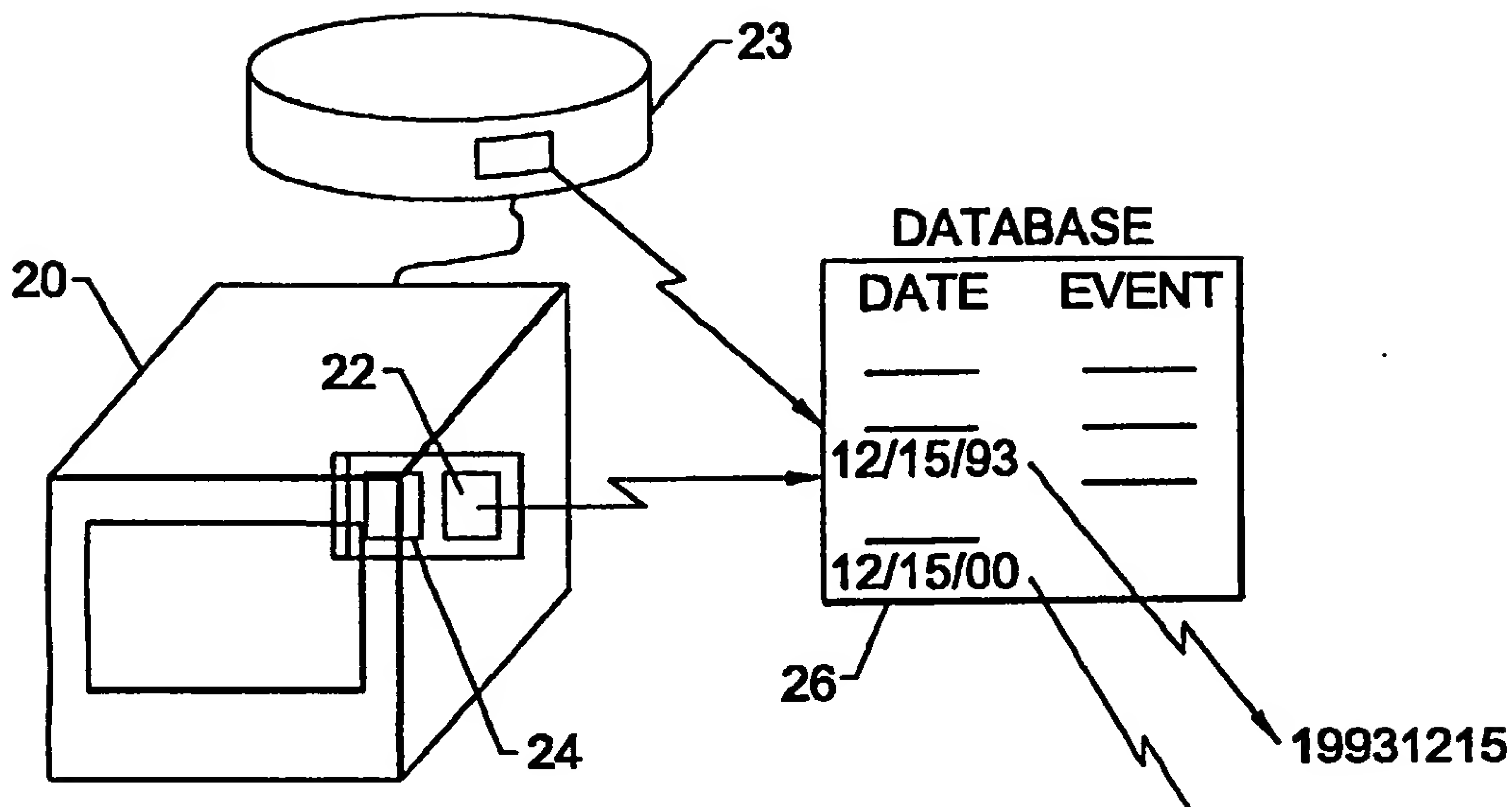

23
DATABASE
DATE EVENT
20
22
12/15/93
12/15/00
26
24
19931215
20001215

FIG. 1.

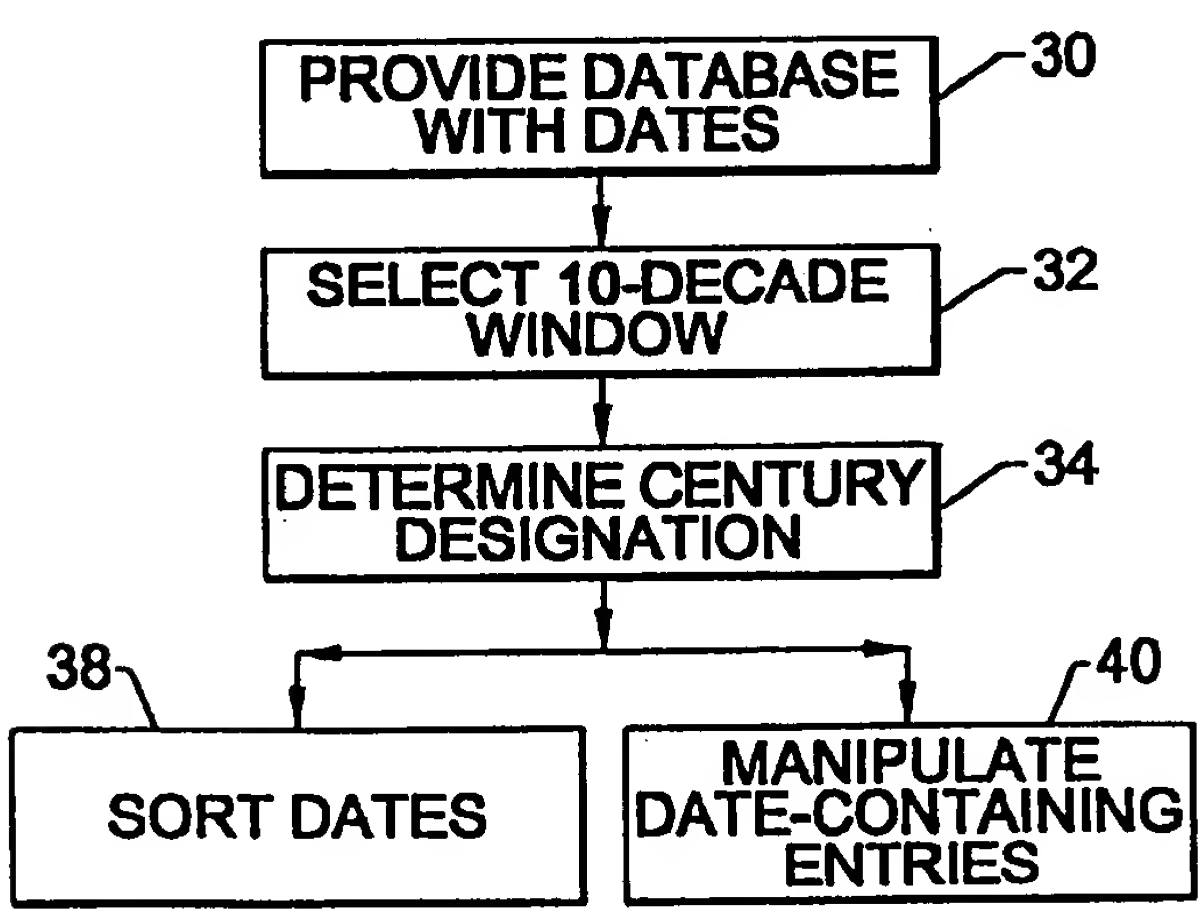

PROVIDE DATABASE WITH DATES
30
SELECT 10-DECADE WINDOW
32
DETERMINE CENTURY DESIGNATION
34
38
40
SORT DATES
MANIPULATE DATE-CONTAINING ENTRIES

FIG. 2.

DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

BACKGROUND OF THE INVENTION

This invention relates to the manipulation of information in a database, and, in particular, to the determination of dates in a useful form.

Dates are stored as symbolic representations in computer databases in varying formats. For example, a date may be represented in the numerical representation MM/DD/YY, where MM is a two-digit month designator, DD is a two-digit day designator, and YY is a two-digit year designator (the last two digits of the year). Thus, Dec. 15, 1993 is designated as 12/15/93. A date may also be represented in an alphanumeric form MMM/DD/YY, where MMM is an alphabetic month designator (e.g., DEC for December), and DD and YY are the same as in the numerical form. Dec. 15, 1993 is represented in this format as DEC/15/93.

Such approaches for the representation of dates have worked well since the advent of computer databases, which has occurred in the twentieth century. Dates may be sorted in chronological order using the numerical representations. However, with the turn of the century at Jan. 1, 2000, the representation and utilization of dates becomes more complex. Using the numerical form above, Dec. 15, 2000 is represented as 12/15/00. If a numerical sort is performed on 12/15/93 and 12/15/00, the later date 12/15/00 sorts as the first-occurring date, an incorrect result.

Sets of dates spanning the turn of the century and associated with past, current, and future activities are now stored in many databases. When stored in the conventional formats discussed above, those dates will not readily be used and numerically sorted in chronological order. They may be manually converted to a more usable form in the sense that programs may be written to perform conversions, manipulations, and sorting. However, these programs typically require additional data fields for storage, which may be objectionable in some circumstances.

There is a need for an improved approach to the representation and utilization of dates in databases, and for converting the existing dates in databases to a more usable form. The present invention fulfills this need, and further provides related advantages.

SUMMARY OF THE INVENTION

The present invention provides an approach to the representation and utilization of dates stored symbolically in databases. Existing symbolic date representations are converted to a more useful form of symbolic date representations without the addition of new data fields, and in a manner that is performed automatically by the computer and requires no user input. The approach of the invention permits direct numerical sorting of dates.

In accordance with the invention, a method of processing dates stored in a database comprises the steps of providing a database with dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the dates falling within a 10-decade period of time. A 10-decade window with a Y_AY_B value for the first year of the ten-decade window is selected, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database. A century designator C_1C_2 is determined for each date in the database, C_1C_2 having a first value if Y_1Y_2 is less

than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B Each date in the database is formatted with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2.

In the case of most practical interest, the 10-decade period of time spans the year 2000 and begins with a year in which the second digit (Y_B in Y_AY_B) is 0 (zero). For any 10-decade period including the year 2000, if the decade designator Y_1 of the date in the database is numerically less than the decade designator Y_A of the first decade of the 10-decade period of time, the century designator C_1C_2 is "20". If Y_1 is equal to or greater than Y_A, C_1C_2 is "19". Dates in databases spanning more than 10 decades are not handled by this approach, but it is not expected that this limitation will be significant for most commercial and industrial databases.

This approach works particularly well if the dates are represented in the format $C_1C_2Y_1Y_2M_1M_2D_1D_2$. The date Dec. 15, 2000 is represented in this format as 20001215, for example. Dates represented in this format may be directly sorted numerically by fast sorting techniques, and thereafter stored back in the database.

The present invention thus provides an efficient approach to converting and utilizing symbolic date representations in databases, which allows automatic processing of dates ranging from before to after the year 2000. The large number of dates represented in some databases may thereby be readily processed and utilized. Other features and advantages of the present invention will be apparent from the following more detailed description of the preferred embodiment, taken in conjunction with the accompanying drawings, which illustrate, by way of example, the principles of the invention. The scope of the invention is not, however, limited to this preferred embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic representation of a computer database with date information therein; and

FIG. 2 is a block flow diagram of a preferred approach for practicing the approach of the invention.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 1 schematically depicts a computer 20 having a read-only or random-access memory 22, a mass-storage device 23, and a central processing unit 24 therein. Stored in the memory 22 or on the mass-storage device 23 is a database 26. The database includes information in the form of symbolic representations of dates and associated information such as events occurring on the respective dates. In a conventional approach, the dates are stored in a format such as $M_1M_2/D_1D_2/Y_1Y_2$ format. M indicates month information, D day information, and Y year information, with the subscript 1 or 2 indicating the first or second digit of the designator, respectively. Dec. 15, 1993 is stored as 12/15/93 or 12-15-93, and Dec. 15, 2000 is stored as 12/15/00 or 12-15-00, for example. If a numerical sort is performed on these dates, 12/15/00 will sort chronologically prior to 12/15/93.

FIG. 2 illustrates the approach of the invention. The computer database 26 is provided, numeral 30, having symbolic representations of dates stored therein. In some cases, the dates will be represented as discussed in the preceding paragraph. In other cases, an alphanumeric designator is used. In that approach, each date is stored as $M_aM_bM_c/D_1D_2/Y_1Y_2$ format, where $M_aM_bM_c$ is an alphabetical symbol such as JAN for January, FEB for February,

etc. In that case, the month designator $M_aM_bM_c$ is first converted to the numerical form M_1M_2 by converting JAN to "01", FEB to "02", etc.

A 10-decade window is selected, numeral 32. That is, it is necessary that all dates in the database will be within some period of 10 decades, or 100 years. This limitation poses little problem for most industrial and commercial databases. The window may be arbitrarily selected. For example, the decade could begin with the 1950's and end with the 2040's, or it could begin with the 1980's and end with the 2070's. The 10-decade window will normally include some decades from the prior century and some from the new century.

The first year of the 10-decade window is represented by Y_AY_B. In a commonly utilized application, Y_B is 0 (zero), although the invention is not limited to this case. That is, the 1950's first decade would be represented by Y_A0 of "50", and the 1980's first decade would be represented by Y_A0 of "80". For this case, a century designator C_1C_2 for a date is determined, numeral 34, by comparing the value of Y_1, the first digit of the year designator for the date, with Y_A, the first digit of the first decade of the 10-decade window. C_1C_2 is assigned a first value if Y_1 is less than Y_A and a second value if Y_1 is equal to or greater than Y_A.

In the case of most interest, the 10-decade window includes decades earlier than the year 2000 and decades later than the year 2000, and Y_B is zero C_1C_2 is assigned "20" if Y_1 is less than Y_A and is assigned "19" if $Y_1$is equal to or greater than Y_A. In that case and for example, if Y_A is 5, meaning that the decade beginning in 1950 was selected as the first decade of the 10-decade window, and if Y_1Y_2 is "43", the century designator C_1C_2 is "20", indicating that the year in question in the database is 2043. On the other hand, if Y_1Y_2 is "63", the century designator C_1C_2 is "19", indicating that the year in question in the database is 1963. This selection process is performed in a completely automated fashion by the computer, without human input other than to select the starting date of the 10-decade window.

The symbolic representations of the dates in the database are reformatted with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2, numeral 36 of FIG. 2. In one case that produces particularly advantageous results for many operations, such as chronological date sorting, the date is represented in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$. For example, the date 12/15/93 (Dec. 15, 1993) is represented as 19931215 and the date 12/15/00 (Dec. 15, 2000) as 20001215. A straightforward numerical sort of date data fields expressed in this form produces an accurate chronological ordering.

Once the symbolic representations of the dates are reformatted according to the procedures set forth above, the date information may be sorted, numeral 38, or otherwise manipulated, numeral 40, together with the entries associated with the dates. Such manipulation may include handling of data associated with the dates, storing the dates and associated information back in the data base, or other processes.

The approach of the invention has been implemented in a computer program, a copy of which is attached as Exhibit A. This program converts dates both before and after the year 2000.

The present invention provides an effective technique for reformatting symbolic representations of date information that is rapid and automated, and yields new symbolic representations of date information that are particularly amenable to further processing. Although a particular embodiment of the invention has been described in detail for purposes of illustration, various modifications and enhance-

ments may be made without departing from the spirit and scope of the invention. Accordingly, the invention is not to be limited except as by the appended claims.

What is claimed is:

1. A method of processing symbolic representations of dates stored in a database, comprising the steps of

providing a database with symbolic representations of dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each symbolic representation of a date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B; and

reformatting the symbolic representation of the date with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2 to facilitate further processing of the dates.

2. The method of claim 1, wherein the 10-decade window includes the decade beginning in the year 2000.

3. The method of claim 2, wherein the step of determining includes the step of

determining the first value as 20 and the second value as 19.

4. The method of claim 1, including an additional step, after the step of reformatting, of

sorting the symbolic representations of dates.

5. The method of claim 1, wherein the step of reformatting includes the step of

reformatting each symbolic representation of a date into the format $C_1C_2Y_1Y_2M_1M_2D_1D_2$.

6. The method of claim 5, including an additional step, after the step of reformatting, of

sorting the symbolic representations of dates using a numerical-order sort.

7. The method of claim 1, wherein the step of providing a database includes the step of

converting pre-existing date information having a different format into the format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator and Y_1Y_2 is the numerical year designator.

8. The method of claim 1, wherein the step of selecting includes the step of

selecting Y_AY_B such that Y_B is 0 (zero).

9. The method of claim 1, including an additional step, after the step of reformatting, of

storing the symbolic representation of dates and their associated information back into the database.

10. The method of claim 9, including the additional step, after the step of reformatting, of

manipulating information in the database having the reformatted date information therein.

11. A method of processing dates in a database, comprising the steps of

providing a database with dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of dates falling within a 10-decade period of time which includes the decade beginning in the year 2000;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B;

reformatting each date in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$ to facilitate further processing of the dates; and

sorting the dates in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$.

12. The method of claim 11, wherein the step of providing a database includes the step of

converting pre-existing date information having a different format into the format wherein M_1M_2 is the numeri-

cal month designator, D_1D_2 is the numerical day designator and Y_1Y_2 is the numerical year designator.

13. The method of claim 11, wherein the step of selecting includes the step of

selecting Y_AY_B such that Y_B is 0 (zero).

14. The method of claim 11, including an additional step, after the step of sorting, of

storing the sorted dates and their associated information back into the database.

15. The method of claim 14, including the additional step, after the step of sorting, of

manipulating information in the database having the reformatted date therein.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,806,063

DATED : September 8, 1998

INVENTOR(S) : Dickens

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On title page, item [56],

In the References Cited, OTHER PUBLICATIONS, line 1, before "The", insert --IBM: --.

In the ABSTRACT, line 4, "$_{M1}M_2$" should read --M_1M_2--.

Signed and Sealed this

Twenty-ninth Day of December, 1998

Attest:

BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

///

90/005592

US005806063A

United States Patent [19]

Dickens

[11] **Patent Number: 5,806,063**

[45] **Date of Patent: Sep. 8, 1998**

[54] **DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY**

[75] Inventor: **Bruce Dickens**, Irvine, Calif.

[73] Assignee: **McDonnell Douglas Corporation**, Long Beach, Calif.

[21] Appl. No.: **725,574**

[22] Filed: **Oct. 3, 1996**

[51] **Int. Cl.**^{6} **G06F 17/30**

[52] **U.S. Cl.** 707/6; 707/102; 707/7; 707/200

[58] **Field of Search** 707/6, 102, 7, 707/200

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,573,127	2/1986	Korff	364/493
5,630,118	5/1997	Shaughnessy	707/1
5,644,762	7/1997	Soeder	707/6
5,668,989	9/1997	Mao	707/101

OTHER PUBLICATIONS

The Year 2000 and 2–Digit Dates: A Guide for Planning and Implementation, Third Edition, May, 1996.

IBM: *The Year 2000 and 2–Digit Dates: A Guide for Planning and Implementation*; First Edition, Oct. 1995.

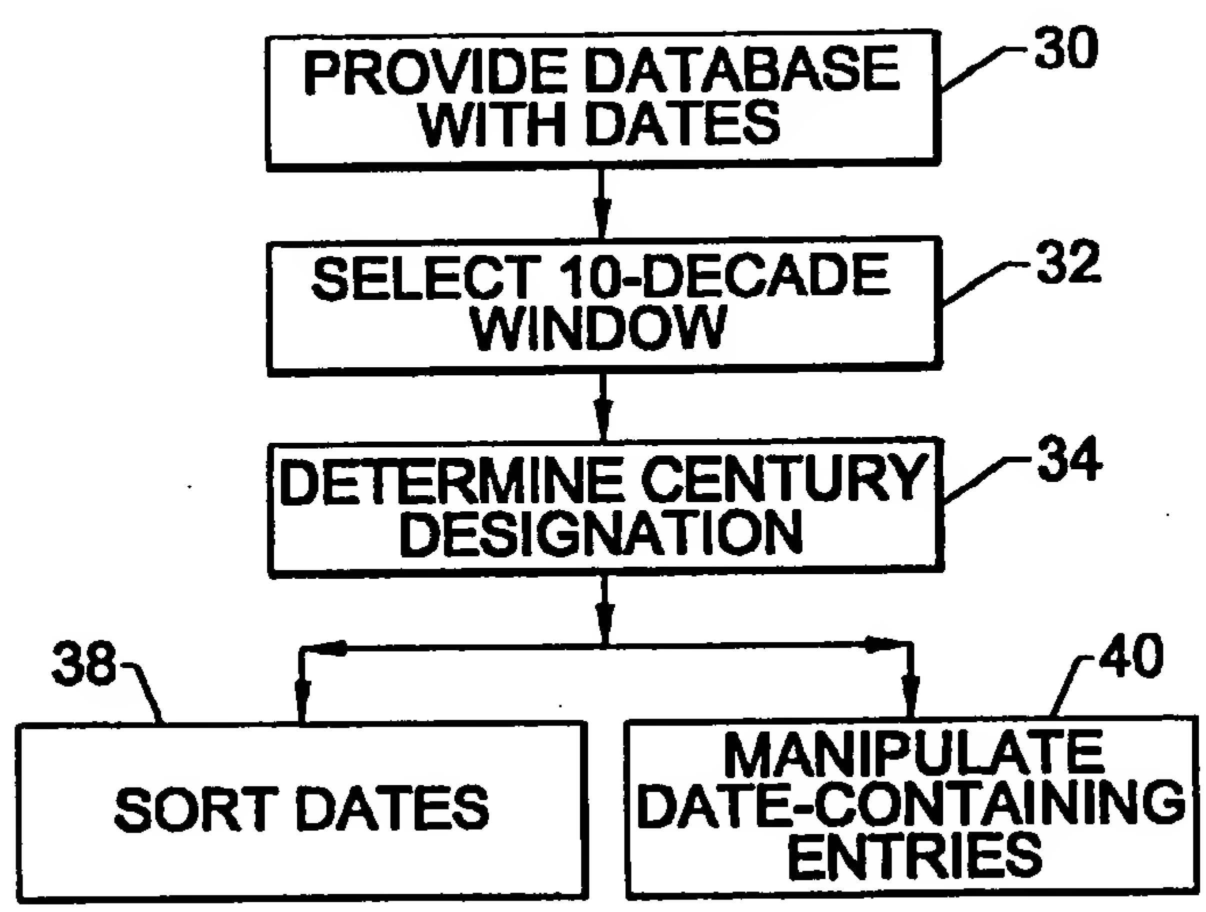
Primary Examiner—Wayne Amsbury

Attorney, Agent, or Firm—Bell Seltzer Intellectual Property Group of Alston & Bird LLP

[57] **ABSTRACT**

Dates stored in symbolic form in a database are reformatted to permit easy manipulation and sorting of date-related information. Each date in M_1M_2, D_1D_2, and Y_1Y_2 format is converted to C_1C_2, Y_1Y_2, $_{M1}M_2$, and D_1D_2 format. To accomplish the conversion, a 10-decade window starting on Y_AY_B is defined that encompasses all dates in the database. The value of C_1C_2 is determined by the relative values of Y_1Y_2 and Y_AY_B. The reformatted date information is particularly useful when the reformatting is in $C_1C_2Y_1Y_2M_1M_2D_1D_2$ format, because sorting by date is accomplished using a pure numerical-value sort.

15 Claims, 1 Drawing Sheet



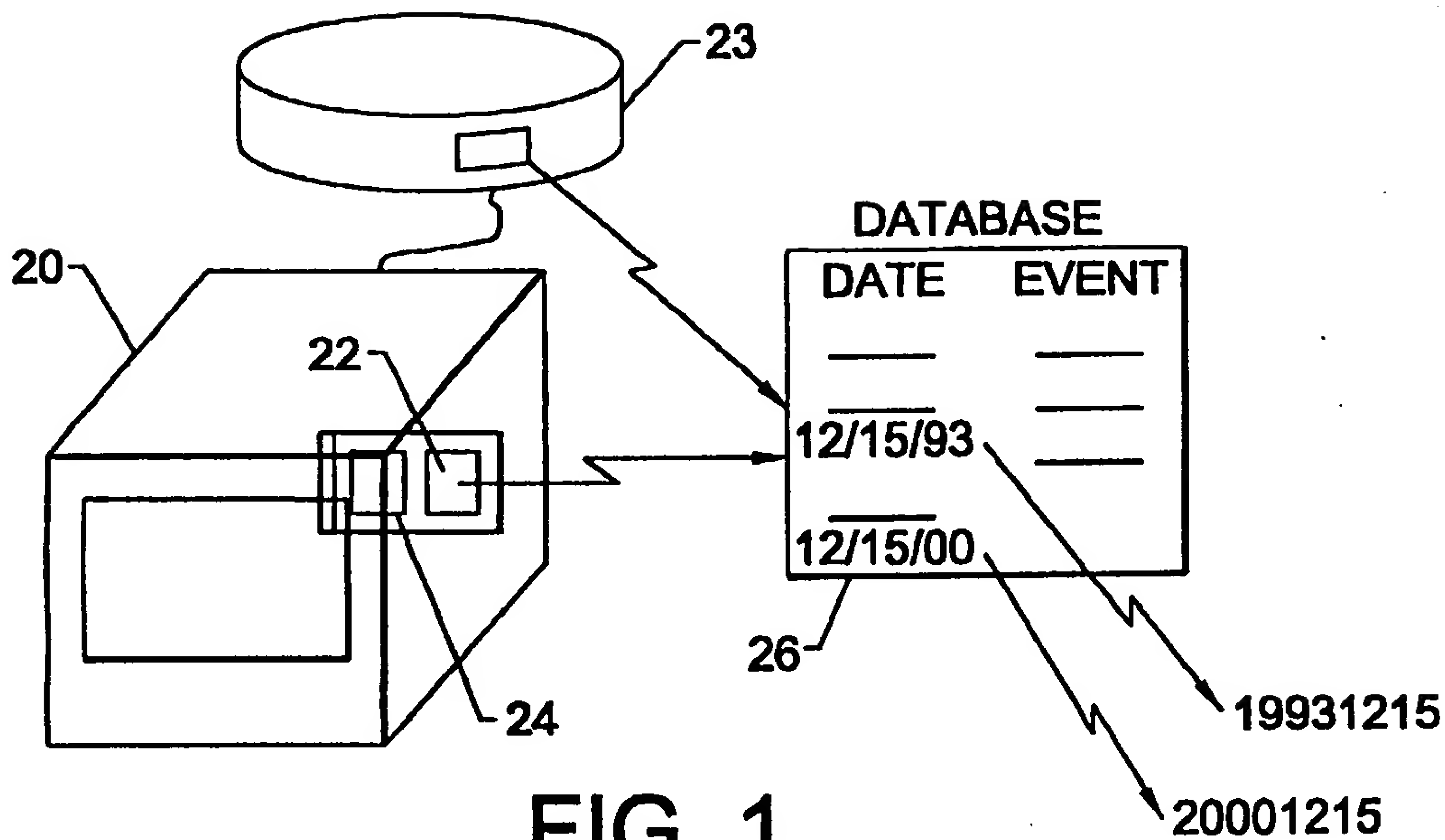

23
DATABASE
DATE EVENT
20
22
12/15/93
12/15/00
26
24
19931215
20001215

FIG. 1.

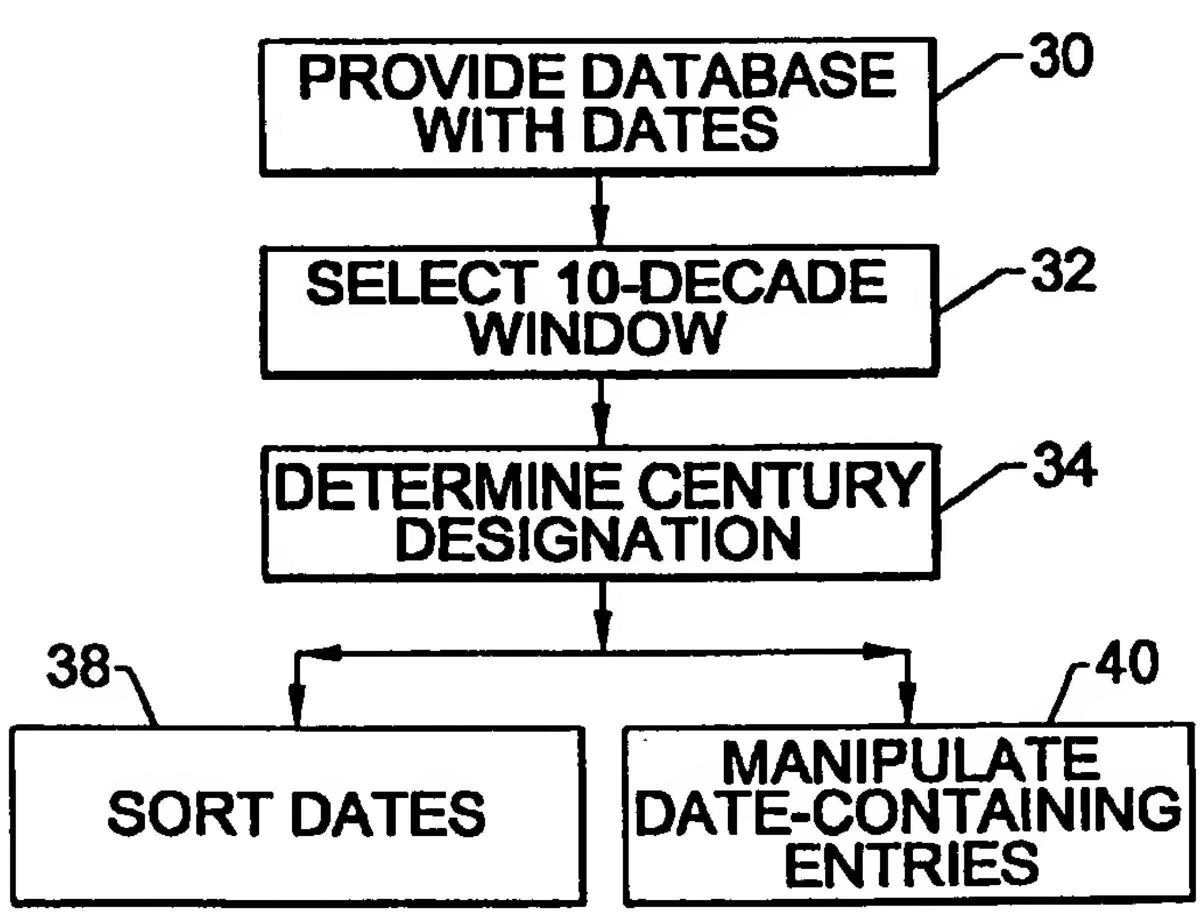

PROVIDE DATABASE WITH DATES
30
SELECT 10-DECADE WINDOW
32
DETERMINE CENTURY DESIGNATION
34
38
40
SORT DATES
MANIPULATE DATE-CONTAINING ENTRIES

FIG. 2.

DATE FORMATTING AND SORTING FOR DATES SPANNING THE TURN OF THE CENTURY

BACKGROUND OF THE INVENTION

This invention relates to the manipulation of information in a database, and, in particular, to the determination of dates in a useful form.

Dates are stored as symbolic representations in computer databases in varying formats. For example, a date may be represented in the numerical representation MM/DD/YY, where MM is a two-digit month designator, DD is a two-digit day designator, and YY is a two-digit year designator (the last two digits of the year). Thus, Dec. 15, 1993 is designated as 12/15/93. A date may also be represented in an alphanumeric form MMM/DD/YY, where MMM is an alphabetic month designator (e.g., DEC for December), and DD and YY are the same as in the numerical form. Dec. 15, 1993 is represented in this format as DEC/15/93.

Such approaches for the representation of dates have worked well since the advent of computer databases, which has occurred in the twentieth century. Dates may be sorted in chronological order using the numerical representations. However, with the turn of the century at Jan. 1, 2000, the representation and utilization of dates becomes more complex. Using the numerical form above, Dec. 15, 2000 is represented as 12/15/00. If a numerical sort is performed on 12/15/93 and 12/15/00, the later date 12/15/00 sorts as the first-occurring date, an incorrect result.

Sets of dates spanning the turn of the century and associated with past, current, and future activities are now stored in many databases. When stored in the conventional formats discussed above, those dates will not readily be used and numerically sorted in chronological order. They may be manually converted to a more usable form in the sense that programs may be written to perform conversions, manipulations, and sorting. However, these programs typically require additional data fields for storage, which may be objectionable in some circumstances.

There is a need for an improved approach to the representation and utilization of dates in databases, and for converting the existing dates in databases to a more usable form. The present invention fulfills this need, and further provides related advantages.

SUMMARY OF THE INVENTION

The present invention provides an approach to the representation and utilization of dates stored symbolically in databases. Existing symbolic date representations are converted to a more useful form of symbolic date representations without the addition of new data fields, and in a manner that is performed automatically by the computer and requires no user input. The approach of the invention permits direct numerical sorting of dates.

In accordance with the invention, a method of processing dates stored in a database comprises the steps of providing a database with dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the dates falling within a 10-decade period of time. A 10-decade window with a Y_AY_B value for the first year of the ten-decade window is selected, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database. A century designator C_1C_2 is determined for each date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B Each date in the database is formatted with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2.

In the case of most practical interest, the 10-decade period of time spans the year 2000 and begins with a year in which the second digit (Y_B in Y_AY_B) is 0 (zero). For any 10-decade period including the year 2000, if the decade designator Y_1 of the date in the database is numerically less than the decade designator Y_A of the first decade of the 10-decade period of time, the century designator C_1C_2 is "20". If Y_1 is equal to or greater than Y_A, C_1C_2 is "19". Dates in databases spanning more than 10 decades are not handled by this approach, but it is not expected that this limitation will be significant for most commercial and industrial databases.

This approach works particularly well if the dates are represented in the format $C_1C_2Y_1Y_2M_1M_2D_1D_2$. The date Dec. 15, 2000 is represented in this format as 20001215, for example. Dates represented in this format may be directly sorted numerically by fast sorting techniques, and thereafter stored back in the database.

The present invention thus provides an efficient approach to converting and utilizing symbolic date representations in databases, which allows automatic processing of dates ranging from before to after the year 2000. The large number of dates represented in some databases may thereby be readily processed and utilized. Other features and advantages of the present invention will be apparent from the following more detailed description of the preferred embodiment, taken in conjunction with the accompanying drawings, which illustrate, by way of example, the principles of the invention. The scope of the invention is not, however, limited to this preferred embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic representation of a computer database with date information therein; and

FIG. 2 is a block flow diagram of a preferred approach for practicing the approach of the invention.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 1 schematically depicts a computer 20 having a read-only or random-access memory 22, a mass-storage device 23, and a central processing unit 24 therein. Stored in the memory 22 or on the mass-storage device 23 is a database 26. The database includes information in the form of symbolic representations of dates and associated information such as events occurring on the respective dates. In a conventional approach, the dates are stored in a format such as $M_1M_2/D_1D_2/Y_1Y_2$ format. M indicates month information, D day information, and Y year information, with the subscript 1 or 2 indicating the first or second digit of the designator, respectively. Dec. 15, 1993 is stored as 12/15/93 or 12-15-93, and Dec. 15, 2000 is stored as 12/15/00 or 12-15-00, for example. If a numerical sort is performed on these dates, 12/15/00 will sort chronologically prior to 12/15/93.

FIG. 2 illustrates the approach of the invention. The computer database 26 is provided, numeral 30, having symbolic representations of dates stored therein. In some cases, the dates will be represented as discussed in the preceding paragraph. In other cases, an alphanumeric designator is used. In that approach, each date is stored as $M_aM_bM_c/D_1D_2/Y_1Y_2$ format, where $M_aM_bM_c$ is an alphabetical symbol such as JAN for January, FEB for February,

etc. In that case, the month designator $M_aM_bM_c$ is first converted to the numerical form M_1M_2 by converting JAN to "01", FEB to "02", etc.

A 10-decade window is selected, numeral 32. That is, it is necessary that all dates in the database will be within some period of 10 decades, or 100 years. This limitation poses little problem for most industrial and commercial databases. The window may be arbitrarily selected. For example, the decade could begin with the 1950's and end with the 2040's, or it could begin with the 1980's and end with the 2070's. The 10-decade window will normally include some decades from the prior century and some from the new century.

The first year of the 10-decade window is represented by Y_AY_B. In a commonly utilized application, Y_B is 0 (zero), although the invention is not limited to this case. That is, the 1950's first decade would be represented by Y_A0 of "50", and the 1980's first decade would be represented by Y_A0 of "80". For this case, a century designator C_1C_2 for a date is determined, numeral 34, by comparing the value of Y_1, the first digit of the year designator for the date, with Y_A, the first digit of the first decade of the 10-decade window. C_1C_2 is assigned a first value if Y_1 is less than Y_A and a second value if Y_1 is equal to or greater than Y_A.

In the case of most interest, the 10-decade window includes decades earlier than the year 2000 and decades later than the year 2000, and Y_B is zero C_1C_2 is assigned "20" if Y_1 is less than Y_A and is assigned "19" if $Y_1$is equal to or greater than Y_A. In that case and for example, if Y_A is 5, meaning that the decade beginning in 1950 was selected as the first decade of the 10-decade window, and if Y_1Y_2 is "43", the century designator C_1C_2 is "20", indicating that the year in question in the database is 2043. On the other hand, if Y_1Y_2 is "63", the century designator C_1C_2 is "19", indicating that the year in question in the database is 1963. This selection process is performed in a completely automated fashion by the computer, without human input other than to select the starting date of the 10-decade window.

The symbolic representations of the dates in the database are reformatted with the values C_1C_2, Y_1Y_2, M_1M_2, and D_1D_2, numeral 36 of FIG. 2. In one case that produces particularly advantageous results for many operations, such as chronological date sorting, the date is represented in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$. For example, the date 12/15/93 (Dec. 15, 1993) is represented as 19931215 and the date 12/15/00 (Dec. 15, 2000) as 20001215. A straightforward numerical sort of date data fields expressed in this form produces an accurate chronological ordering.

Once the symbolic representations of the dates are reformatted according to the procedures set forth above, the date information may be sorted, numeral 38, or otherwise manipulated, numeral 40, together with the entries associated with the dates. Such manipulation may include handling of data associated with the dates, storing the dates and associated information back in the data base, or other processes.

The approach of the invention has been implemented in a computer program, a copy of which is attached as Exhibit A. This program converts dates both before and after the year 2000.

The present invention provides an effective technique for reformatting symbolic representations of date information that is rapid and automated, and yields new symbolic representations of date information that are particularly amenable to further processing. Although a particular embodiment of the invention has been described in detail for purposes of illustration, various modifications and enhancements may be made without departing from the spirit and scope of the invention. Accordingly, the invention is not to be limited except as by the appended claims.

What is claimed is:

1. A method of processing symbolic representations of dates stored in a database, comprising the steps of

providing a database with symbolic representations of dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of the symbolic representations of dates falling within a 10-decade period of time;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each symbolic representation of a date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B; and

reformatting the symbolic representation of the date with the values C_1C_2, Y_1Y_2, M_1M_2 , and D_1D_2 to facilitate further processing of the dates.

2. The method of claim 1, wherein the 10-decade window includes the decade beginning in the year 2000.

3. The method of claim 2, wherein the step of determining includes the step of

determining the first value as 20 and the second value as 19.

4. The method of claim 1, including an additional step, after the step of reformatting, of

sorting the symbolic representations of dates.

5. The method of claim 1, wherein the step of reformatting includes the step of

reformatting each symbolic representation of a date into the format $C_1C_2Y_1Y_2M_1M_2D_1D_2$.

6. The method of claim 5, including an additional step, after the step of reformatting, of

sorting the symbolic representations of dates using a numerical-order sort.

7. The method of claim 1, wherein the step of providing a database includes the step of

converting pre-existing date information having a different format into the format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator and Y_1Y_2 is the numerical year designator.

8. The method of claim 1, wherein the step of selecting includes the step of

selecting Y_AY_B such that Y_B is 0 (zero).

9. The method of claim 1, including an additional step, after the step of reformatting, of

storing the symbolic representation of dates and their associated information back into the database.

10. The method of claim 9, including the additional step, after the step of reformatting, of

manipulating information in the database having the reformatted date information therein.

11. A method of processing dates in a database, comprising the steps of

providing a database with dates stored therein according to a format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator, and Y_1Y_2 is the numerical year designator, all of dates falling within a 10-decade period of time which includes the decade beginning in the year 2000;

selecting a 10-decade window with a Y_AY_B value for the first decade of the window, Y_AY_B being no later than the earliest Y_1Y_2 year designator in the database;

determining a century designator C_1C_2 for each date in the database, C_1C_2 having a first value if Y_1Y_2 is less than Y_AY_B and having a second value if Y_1Y_2 is equal to or greater than Y_AY_B;

reformatting each date in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$ to facilitate further processing of the dates; and

sorting the dates in the form $C_1C_2Y_1Y_2M_1M_2D_1D_2$.

12. The method of claim **11**, wherein the step of providing a database includes the step of

converting pre-existing date information having a different format into the format wherein M_1M_2 is the numerical month designator, D_1D_2 is the numerical day designator and Y_1Y_2 is the numerical year designator.

13. The method of claim **11**, wherein the step of selecting includes the step of

selecting Y_AY_B such that Y_B is 0 (zero).

14. The method of claim **11**, including an additional step, after the step of sorting, of

storing the sorted dates and their associated information back into the database.

15. The method of claim **14**, including the additional step, after the step of sorting, of

manipulating information in the database having the reformatted date therein.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,806,063

DATED : September 8, 1998

INVENTOR(S) : Dickens

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On title page, item [56],

In the References Cited, OTHER PUBLICATIONS, line 1, before "The", insert --IBM: --.

In the ABSTRACT, line 4, "$_{M1}M_2$" should read --M_1M_2--.

Signed and Sealed this

Twenty-ninth Day of December, 1998

Attest:

Bruce Lehman

BRUCE LEHMAN

Attesting Officer *Commissioner of Patents and Trademarks*

Ohms

Computer processing of dates outside the twentieth century

by B. G. Ohms

This paper presents practical solutions to problems envisioned in extending computer processing of dates beyond the twentieth century. Many data processing managers are concerned with processing cross-century dates, and in doing so using existing systems, with a minimum of disruption to normal operations. The use of existing date formats can eliminate the need for massive system modifications. Methods of using existing date formats across century boundaries are explained. The use of a format termed the Lilian date format in honor of Luigi Lilio, the inventor of the Gregorian calendar, is introduced. The requirements for an effective date-processing algorithm are presented.

The Gregorian calendar serves us quite well in our day-to-day living. Due to discontinuities in various date divisions, however, it is not readily adaptable to computer programming. This fact becomes more apparent as we approach the new century. Few efficient, easy-to-use functions for manipulating dates have been produced. Also to be considered are the human requirements for ease of use, development, and maintenance. Other considerations include storage costs, efficiency, and adaptability across many different applications and environments.

Some early date-conversion programs were acts of expediency rather than planning, created to solve specific problems rather than for general programming use. Different functions were created at different times. Naming conventions, invocation formats, and implementation methods have often been inconsistent. Programs that provided a day-of-week function were rare. Also rare were programs for deriving a new date by adding or subtracting from another date in a different year. The functions that were available were normally not part of an integrated system for providing compatibility with most of the common date formats. Since documentation was not always created or maintained, individuals were often unaware of what was available. Today, dating format standards are being discussed internationally. The date-processing method presented in this paper is expected to be compatible with any foreseeable international standard date format as well as with the several formats discussed in this paper.

Typically, dates are displayed and stored in the *Julian* and *Gregorian* formats. Concepts and formats of both Julian and Gregorian date formats are discussed later in this paper. Simply stated, the Julian date is the number of the year together with the serial number of the day of that year. The Gregorian date format consists of month, day of month, and year. Many calendars show Julian dates printed in small numerals.

A format termed the *Lilian date format* is presented here as the basis for making date conversions of the

© Copyright 1986 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

type mentioned earlier. The Lilian format, which may be stored in three bytes of memory, provides the storage capacity for dates well beyond the year 10 000. This format handles processing across century years and other aspects of date conversion not currently adaptable to computer programming. Practical date processing services must provide ease of use for the end user, developer, and maintainer. Such services must also eliminate date ambiguity, achieve excellent performance, and minimize storage.

The two positions traditionally used in both Julian and Gregorian date formats implicitly represent a year within a century. However, this system is inadequate for representing dates in more than one century. For example, it is ambiguous as to whether 03 represents the year 1903 or the year 2003. The Lilian date format avoids the ambiguity by using seven positions for the number of days from the beginning of the Gregorian calendar, October 15, 1582.

Origins of date complexity

Julian calendar. The Julian calendar was established in 46 B.C. by Julius Caesar.[1] It replaced a system of constant adjustments, more for political reasons than for correcting inaccuracies in timekeeping. After a bad start, with the first few leap years being calculated incorrectly, it served quite adequately for many centuries. The Julian calendar is not the same as the Julian date, which was previously defined. The Julian calendar was a time-measuring system, which we now discuss.

The scheme of the Julian calendar was quite simple. The calendar year consisted of 365-day years, with years evenly divisible by four having 366 days. The resulting average year of 365.25 days was slightly longer than the *tropical year*, which is based on the time marked by the passage of equinoxes. That slight difference resulted in a gradual drift of the calendar year. The resulting difficulty in properly setting the date of celebration of Easter caused great concern to the Roman Church. The date for Easter is related to the date for Passover, which is related to the vernal equinox. By the sixteenth century, the discrepancy approximated ten days, and the desire for calendar reform intensified. An artifact of this discrepancy manifests itself today in the differences in dates for Christmas and Easter between the Western Church and the Eastern Church. The latter continues to use the Julian calendar dates as they were at the time of the Gregorian calendar reform. That is, the Eastern Church recognizes the Gregorian calendar, but for certain feast days does not void the ten-day error that existed at the time of the calendar reform.

Gregorian calendar. For the calendar reform, Pope Gregory XIII selected[2] the plan of Aloysius Lilius

England and her colonies adopted the Gregorian calendar on Thursday, September 14, 1752.

(1510–1576?),[1] who is also known as Luigi Lilio.[3] This reform, known as the Gregorian calendar, was implemented on Friday, October 15, 1582.[2]

Although use of the Gregorian calendar spread rapidly among Roman Catholic countries, many centuries passed before it was generally used by non-Catholic countries. England and her colonies, including what is now the United States, adopted the calendar on Thursday, September 14, 1752.[3] Many other countries—notably China, Greece, and Russia—did not adopt it until after the beginning of the 20th century. In fact, Russia adopted the Gregorian calendar on two separate occasions, first in 1918, about the same time as many other countries, and again on June 27, 1940.[4] Changes made in Russia in 1929 to avoid the religious associations of the Gregorian calendar were unsuccessful, and it was reimplemented in 1940.

The reform that synchronized the calendar year with the tropical year required the removal of ten days from the calendar. As a result, Friday, October 15, 1582, immediately followed Thursday, October 4, 1582. (An alternate plan would have removed the ten excess days gradually by canceling leap days for 40 years.) As before, every fourth year is a leap year, and, to maintain synchronization, centurial years that are evenly divisible by 400 are also leap years.[2] Thus, for example, 1900 was a common year; the year 2000 will be a leap year, and the year 2100 will start a series of common centurial years again. The result is an average calendar year of 365.2425 days, which closely approximates the tropical year. Despite

this close approximation, by the 44th century,[5] the Gregorian calendar will again differ from the tropical year by one day. Because the tropical year consists of an irrational number of days, no calendar year with an integral number of days can exactly match the tropical year. Not only is there not an integral

Discontinuities in the Gregorian calendar scheme cause most of the difficulties with date manipulation.

number of days in a tropical year, but also the length of the day is not constant. That is, the length of the tropical year is also changing gradually.

Algorithms for date processing

Centurian date format. Discontinuities in the Gregorian calendar scheme cause most, if not all, of the difficulties associated with date manipulation. Instances of discontinuities are the following:

- No calendar unit is evenly divisible by weeks.
- The number of days per month varies.
- The number of days per year varies.
- The number of days per century varies.

These facts must be accounted for in calculating the number of days between two dates, calculating a date based on the addition or subtraction of days from another date, and calculating the day of the week for a date. A *centurian date format* (DDDDD, representing the day within a century) works well by giving continuous values within a century. Also, other date formats and day of week are readily calculable from this value. The centurian format is limited to a century and results in discrepancies when a century boundary is crossed. Consideration must be given to century years when making a leap year calculation. The DDDDD format will not span more than 273 years (or 179 years in a two-byte binary format). Also, a standard point of origin for the starting day must be defined.

Lilian date format. The Lilian date format consists of seven positions for the number of days from the beginning of the Gregorian calendar. Day one is Friday, October 15, 1582, and the value is incremented by one for each subsequent day. Based on this format are services supporting 44 experimental functions (or more if the three DMY, MDY, and YMD experimental versions of the Gregorian format are counted) synthesized from eight mnemonics.

Function-naming convention. Functions are named to promote easy recall and to suggest the activity to be performed. In the algorithm and experimental PL/I program discussed in this paper, each function name is synthesized from two 3-letter mnemonic parts. These mnemonic parts are defined as follows:

- CLL: compact Lilian date
- DAY: day of the week
- GRG: Gregorian date
- JUL: Julian date
- LIL: Lilian date
- SGR: short Gregorian date
- SJL: short Julian date
- VAL: validate a date

The first part is a description of the *target*, and the second part is a description of the *source*. VAL (validation) and DAY (day of week) can appear only in the target position of the function.

Arguments presented to any of the conversion functions are always presented in the same order: new date (target); old date (source); range date (control), when required; and Gregorian format [specifier(s)], when required. Table 1 illustrates the format for conversion arguments. In all instances, a return code is set.

The format descriptions in Table 1 refer to the type of data—D for day, M for month, and Y for year, as well as the number of positions (e.g., YY for two year positions)—that the data occupy. The origin of the term "Julian date" for the YYDDD format is given in Reference 6. Nevertheless, dates represented in the Julian format conform to the Gregorian calendar and not to the Julian calendar. The Julian date for Thursday, November 14, 1985, is 85318 (short form) and 1985318 (extended form).

Algorithms. The process of conversion between a Lilian format date and a Julian format date is now described. The algorithms are simpler to calculate by choosing a virtual base date rather than the real one. After the calculations are made, any discrepancies are resolved. In the following algorithms, the num-

Table 1 Examples of conversion function arguments

Argument	Description
target=JULSJL (source, 1925)	Conversion to a Julian (YYYYDDD) date from a short Julian (YYDDD) date within the range 1925–2024
target=JULGRG (source, DMY)	Conversion to a Julian (YYYYDDD) date from a Gregorian (DDMMYYYY) date
target=LILGRG (source, YMD)	Conversion of a Lilian (packed format) date from a Gregorian (YYYYMMDD) date
target=CLLJUL (source)	Conversion of a compact Lilian (binary format) date from a Julian date
target=SGRGRG (source, YMD, MDY)	Conversion of a short Gregorian (YYMMDD) date from a Gregorian (MMDDYYYY) date
CALL VALJUL (source)	Validation of a Julian date
target=DAYSJL (source 1925)	Day of week for a short Julian date

bers in parentheses are results of calculations made on the previously given date, November 14, 1985.

Extended Julian to Lilian. Given an extended Julian date (YYYYDDD), compute the corresponding Lilian date. First, compute the number of days from virtual January 1, 1501, to the start of the year being converted (1985318 Julian).

- Subtract 1501 from the Julian year (484).
- Multiply the difference by 365.25 (i.e., the average number of days per year) (176781.00).
- Truncate the result. The leap day is kept only for full four years (176781).

Then compute the number of Julian calendar (not Julian format) days from October 15, 1582, to the date being converted.

- Subtract 29872, i.e., the number of days between virtual January 1, 1501, and October 15, 1582 (146909).
- Add the Julian days (+318 = 147227).

Next, make the Gregorian calendar adjustments to this value.

- Subtract 1501 from the Julian year (484).
- Divide the difference by 100. One leap year is usually skipped each century (4.84).
- Truncate the result to obtain the number of whole leap days. Partial leap days do not exist (4).
- Subtract the number of whole leap days from the number of Julian calendar days (147223).

Because one out of every four centuries keeps all of its leap years, use the virtual year 1201, which is the beginning of the four-century cycle that contains the year 1582, to compute the number of leap years up to the target date.

- Subtract 1201 from the Julian year. The first four-hundred-year cycle began with virtual year 1201 and ended in the real 1600 (784).
- Divide the difference by 400 because one century leap year per 400 years is kept (1.96).
- Truncate the result because partial leap days do not exist (1).
- Add these leap days back into the adjusted Julian calendar days (147224).

This final result is the number of Gregorian calendar days from the beginning of the Gregorian calendar (October 15, 1582) to the date being converted. This result is the sought-for Lilian date.

Lilian to extended Julian. The purpose of this example is to show the procedure for converting a Lilian date to an extended Julian date. First, create a virtual Lilian date, with a starting point of January 1, 1201. Convert this result into a Julian calendar (not Julian format) date. Then convert the Julian calendar date to a Julian format date. The procedure is as follows (147224 Lilian):

- Add 139444 to the Lilian date. This is the number of days from virtual January 1, 1201 (the start of a 400-year cycle) to October 15, 1582. The result is a pseudo-Lilian date (286668).
- Divide the pseudo-Lilian date by 36524.25, the average number of days per century (7.85).
- Truncate the result to obtain the number of century leap days (7).

- Add the number of century leap days to the pseudo-Lilian date (286675).
- Divide the number of century leap days by 4 (1.75).
- Truncate the result to obtain the number of four-century leap days (1).
- Subtract the number of four-century leap days from the pseudo-Lilian date, because they are already included (286674).

The result is the number of full Julian calendar days from January 1, 1201, to the date being converted. We now convert this to an extended Julian format date.

- Divide full Julian calendar days by 365.25. This usually gives one less than the year for the date being converted (784.87).
- If there is no remainder from the division, subtract one from the quotient; otherwise, truncate the quotient to get the pseudo-Julian prior year (784).
- Multiply the pseudo-Julian prior year by 365.25 to determine the number of annual Julian calendar days prior to the year for the date being converted (286356.00).
- Truncate the number of annual Julian calendar days to eliminate partial leap days (286356).
- Subtract the number of truncated annual Julian calendar days from the full number of Julian calendar days to obtain the number of current Julian calendar days in the year of the date being converted (318).
- Add 1201 to the pseudo-Julian prior year to obtain the real Julian year, i.e., 1200 for years prior to 1200 plus one for the current year (+784 = 1985).
- Multiply the real Julian year by 1000 to put it into its proper Julian format position (1985000).
- Add the current Julian calendar days to the result to complete the Julian format date from the Lilian format date (1985318).

Ease of conversion

Accommodating end users. End users usually enter two digits for the year in a date and understand the ambiguity that this represents. Therefore, even at the turn of the century, to avoid adverse user reaction, programs must continue to function with only two digits for year. The inference of the year 1997 from 97 and 2003 from 03 must continue. For the exceptional case where the correct meaning could be 1897 and 1903, entry of all four digits may be required.

The month-day-year and day-month-year formats are ambiguous. Therefore, it might be advisable to continue presenting the date in its conventional U.S. format with a parenthetical explanation of the format—such as (MM/DD/YY)—to avoid the ambiguity.

It is not necessary to change date formats in files, because it is possible to change the programs only.

However, it may be necessary to provide a conversion function that receives a definition of the implied century as a parameter. An excellent way to do this unambiguously is to specify a year as the desired starting point of a 100-year range. For example, if the starting year for the range is specified as 1925, dates with year digits of 25 through 99 would be between 1925 and 1999, and dates with year digits of 00 through 24 would lie between 2000 and 2024.

Accommodating systems support. The conversion of isolated files to new date formats presents a rather trivial problem. In most cases, however, it is not possible to isolate the process. All programs that access the modified data must be changed simultaneously. In some large systems, literally thousands of programs may be involved. In these large systems, it may be prudent to avoid the cost and risk of massive changes in a short period of time.

It is not necessary to change date formats in files, because it is possible to change the programs only, so that the implied century in a date is recognized. Of course, in the vast majority of cases, that is exactly what does take place. Dates familiarly and implicitly exist within the 100-year range beginning with 1900 or 1901. Thus it is necessary merely to modify the programs so that the 100-year range starts at a later date. A beginning date set eighty years prior to the current systems date may be a reasonable convention. This is well within the range now in use.

The significant feature of this approach is that everything need not be modified at once. The modifications may be made over a period of years during

normal program maintenance. Of course, as systems are maintained or replaced, it would be practical to implement full information date formats. Where systems contain dates that span a range of more than 100 years, the century must already have been carried. In the rare event that this is not true, immediate conversion is unavoidable. Fortunately, in most applications, we can deal with centuries as exceptions rather than as a common problem.

Computational considerations

Storage. The two main considerations pertaining to storage are (1) the cost of storing large quantities of data; and (2) the computational cost of converting records within computer files to larger date-field sizes. When millions of dates are stored, as they are in most business systems, every additional byte required to save a single date multiplies to millions of additional bytes of storage. The programming necessary to accommodate larger date-field sizes in records further complicates date conversion.

Putting bounds on data-storage costs at reasonable levels and reducing conversion complications are both achievable. The allocation of additional space to record four positions for year, rather than the traditional two, is not the only possibility. Many systems currently store dates internally in packed Julian format, requiring three bytes of storage. In packed format, a Lilian date or an extended Julian date (YYYYDDD) both require four bytes of storage. However, if a binary format were to be adopted, either form of date could be stored in the three bytes used at present.

A more restrictive situation exists for systems in which dates are stored in two bytes instead of three. These systems use a binary format to record centurian or other forms of dates. In the near term for these systems, it may be necessary to continue storing dates in only two bytes. This cannot be accommodated with a Julian format. However, it is possible to store a range of dates by taking advantage of the continuous characteristic of a Lilian date.

Using the Lilian date system, any date in a selected range of 179 years may be stored as follows. Assume that the selected range is January 1, 1901, through December 31, 2079. Find the Lilian value of December 31, 1900. Subtract this value from the Lilian value of the date to be stored. Convert the result to a 16-bit (two-byte) binary value and store the result. Reverse the process to restore the two-byte date to Lilian format. Continuing to store dates in two bytes should be considered only where the programming cost of increasing record sizes in an existing system is prohibitive. The decreasing cost of storage makes the use of the full Lilian or Julian format a practical possibility when an application is being modified.

In the experiments on which this paper is based, the three-byte Lilian format for data storage has been found to be preferable. Internally, in computer use, the continuous nature of the Lilian date accommodates the types of processing that normally take place within a program. In addition, for all three storage sizes, a consistent format (i.e., the all-Lilian format or the quasi-Lilian format) is maintained.

The continuous nature of the Lilian date accommodates the types of processing that normally take place.

Flexibility. In our experiments, the IBM System 360/370 assembly language was used for coding the date functions. However, the method is easily adaptable to other programming languages such as COBOL, PL/I, and RPG. The experimental functions were also made re-entrant for flexibility within on-line environments.

Validity. Ideally, the validation of a date is necessary only at the point of its manual entry into a system. Experience teaches us that it is not unusual for an interfacing system to provide invalid dates. Therefore, all dates passed to conversion functions must be validated. When an invalid date is encountered by the experimental system, a null value is returned to the invoking program and a return code is set to indicate the nature of the invalid condition.

Efficiency. Date conversions are used heavily within certain applications. Because of the impact on a single system or an installation, efficiency must be inherent in date-conversion programs. Storage requirements for external display and internal calculations by computer programs are expected to mo-

tivate a high volume of conversions. Widely used programs in high-volume environments must perform well and not consume excessive amounts of storage. A date-conversion program that is good in all other ways will not be widely used if it is an operational bottleneck.

For the program discussed in this paper, written in System 360/370 assembly language and implemented as a set of PL/I functions, the experimental results indicated good efficiency. The longest execution paths, including PL/I prologue, validation, conversion, and PL/I epilogue, are a little more than one hundred machine-language instructions. Although the functions appear to the invoking PL/I program to be separate programs, they are all actually provided within a single program that requires less than 4K bytes of storage.

Experimental results indicated good efficiency.

Concluding remarks

Programmers require date-processing functions that effectively handle applications for both the present and the future. For the present, it is sufficient to validate source dates, to convert from one traditional date format to another, and to perform addition or subtraction operations involving dates. For the future, additional functions are required that support processing restricted only by the limitations of the Gregorian calendar. These functions must be fully compatible with existing date-format and record-size restrictions. Massive conversion efforts should not be required to process and store dates outside the twentieth century. Also, end users should be able to continue to use existing two-digit date formats when interacting with computer systems. In all cases the programs that provide these services must be reliable, efficient in the use of both processor and storage, and flexible in application.

Programs that embody all these qualities have been written and tested experimentally. The application as written requires less than 4K bytes of storage and has an average execution path of fewer than 100 machine language instructions. Re-entrancy and the possibility of multilanguage implementation indicate excellent flexibility. This approach presents a practical method of processing dates that is compatible with any dating format standard.

Acknowledgments

No significant work is done in isolation. Over the years, I have experienced the inspiration, assistance, and patience of many individuals. My colleagues Tom Gauthier and Jim Willard first sparked my interest in date processing several years ago. Recently Alex Chang, Barb Henderson, Jack Henriksen, Fred Lange, Bob Lord, Libby Ross, Karen Seabury, and Billy Shih have patiently served as a sounding board, made helpful suggestions, and have been active supporters. Sandy Mink provided valuable editorial support. Many unnamed others have been involved in my experiment, including every member of my family. Their support is also greatly appreciated.

Cited references

1. G. Moyer, "Luigi Lilio and the Gregorian reform of the calendar," *Sky and Telescope* 64, No. 11, 418–419 (November 1982).
2. J. J. Bond, *Handy Book of Rules and Tables for Verifying Dates Within the Christian Era*, Russell & Russell, a Division of Atheneum House, Inc., New York (1966).
3. *The New Encyclopedia Britannica*, Encyclopedia Britannica, Inc., Chicago (1980), p. 602.
4. F. Parise, Editor, *The Book of Calendars*, Facts on Life, Inc., New York (1982).
5. G. Moyer, "The Gregorian calendar," *Scientific American* 246, No. 5, 144–152 (May 1982).
6. M. A. Covington, "A calendar for the ages," *PC Tech Journal* 3, No. 12, 136–142 (December 1985). Joseph Justice Saliger (1540–1609) named the Julian date in honor of his uncle Julius.

General references

G. Moyer, "Astronomical scrapbook—Notes on the Gregorian calendar reform," *Sky and Telescope* 64, No. 12, 530–533 (December 1982).

International Organization for Standardization, "Writing of calendar dates in all-numeric form," Reference No. ISO 2014-1976(E) (April 1976).

International Organization for Standardization, "Information processing interchange—Representation of ordinal dates," Reference No. ISO 2711-1973(E) (January 1973).

Bruce G. Ohms *IBM Information Systems Group, 301 Merritt 7, Norwalk, Connecticut 06856.* Mr. Ohms joined IBM in 1967 and is currently a senior programmer/analyst working in the area of applications systems development. Prior to his current assignment, he has held a variety of positions in programming, systems design, analysis, and development center implementations. He received an A.A.S. degree in data processing from Belleville Area College, Belleville, Illinois, and he attended Yale University.

Reprint Order No. G321-5274.